AMIGA
DEVELOPERS CONFERENCE NOTES

MILAN, 1991

CONFIDENTIAL

The Amiga Developers Conference Notes are CONFIDENTIAL. THE INFORMATION CONTAINED HEREIN IS PRELIMINARY.

Amiga Developers Conference Notes

Milan, 1991

Copyright

Copyright 1991 Commodore-Amiga. Inc. All rights reserved. Some of the materials used herein have been reproduced with the permission of the authors indicated. Use or reproduction of such materials shall be in accordance with the authors' instructions. Commodore and Amiga are registered trademarks of Commodore Electronics Ltd. and Commodore-Amiga, Inc. respectively. This document may also contain reference to other trademarks and registered trademarks for the various products listed which are believed to belong to the sources associated therewith.

Warning

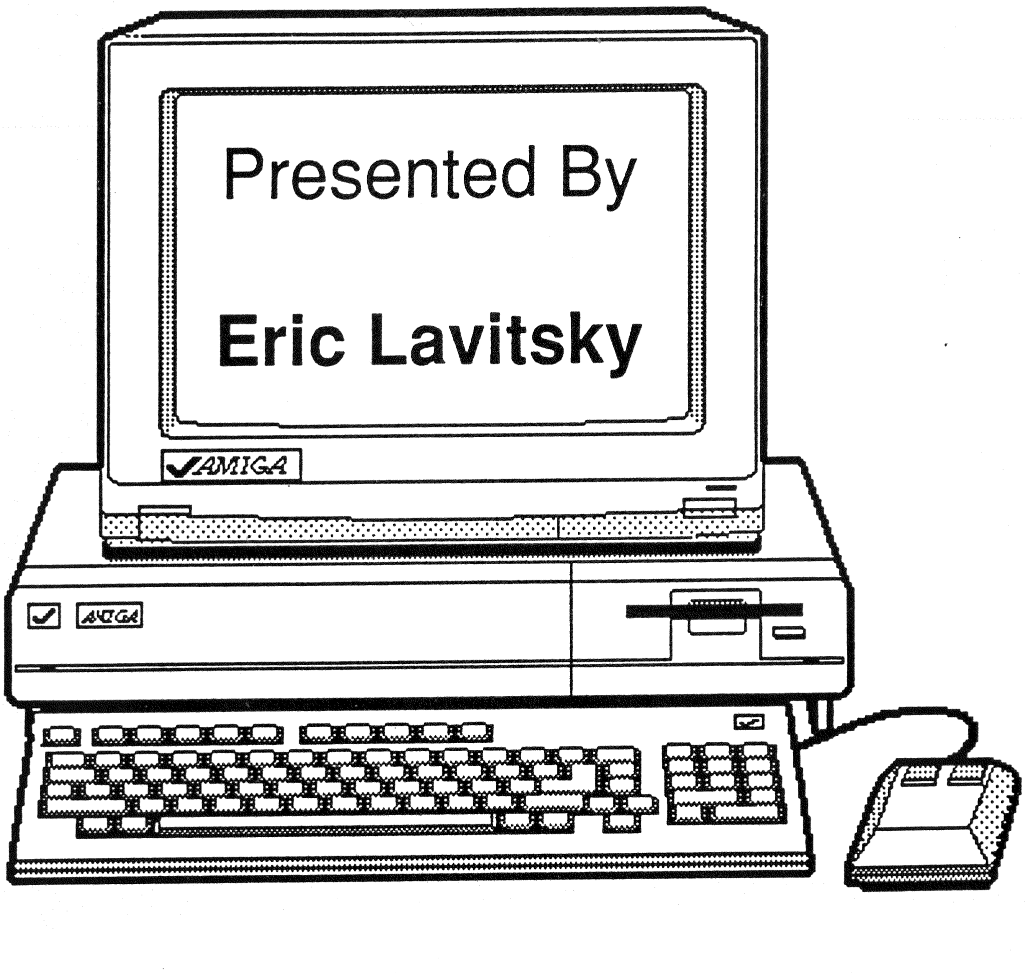
The information contained herein is subject to change without notice. Commodore specifically does not make any endorsement or representation with respect to the use, results, or performance of the information (including without limitation its capabilities, appropriateness, reliability, currentness or availability).

Disclaimer

This information is provided "as is" without warranty of any kind, either express or implied. The entire risk as to the use of this information is assumed by the user. In no event will Commodore or its affiliated companies be liable for any damages, direct, indirect, incidental, special or consequential, resulting from any defect in the information, even if advised of the possibility of such damages.

Introduction To Programming The Amiga®

Part I



Presented By

Eric Lavitsky

Copyright © 1988-1991 by Eric Lavitsky
All Rights Reserved Worldwide

This course is the exclusive property of the author,
Eric Lavitsky, 174 Hyde Park Rd., Somerset, NJ 08873 (908) 560-0106
and may not be reproduced in any way without express written permission.

This course is dedicated to all the wonderful friends and accquaintances I have made over the past few years working with the Amiga and to everyone who made the Amiga a reality.

Amiga is a registered trademark of Commodore-Amiga Inc.

This document was created entirely on an Amiga A2000/A2500 using Professional Page from The Gold Disk Inc. and printed on a PostScript Laser Printer.

What Are The Goals Of This Course?

- To provide an understanding of fundamental concepts of Amiga architecture and programming.
- To provide the basic tools to begin writing Amiga applications.
- To provide insight into elements of good (and bad) Amiga programming style.

Prerequisites:

This course assumes a knowledge of ‘C’ or a similiar structured programming language and an understanding of basic concepts in computer architecture. A familiarity with assembly language and other multi-tasking/multi-processor environments is not required, but will be helpful.

Disclaimer: Many of the examples in this course are presented as fragments or pseudo-code. They were derived from various C Compilers/Assemblers and tested wherever possible. The author has made every attempt to verify their accuracy, but assumes no liability for their content.

Course Outline

Overview of Amiga Architecture

- **Hardware**
  - CPU, Custom Chips, RAM/ROM, Expansion
- **ROM Kernel**
  - Libraries, Devices, Hierarchy

Starting Out With EXEC

- **Library Access**
  - Libray Base Address and Offsets
  - Using The Autodocs
  - Using OpenLibrary/CloseLibrary
  - System Library Summary
- **Lists**
  - Data Structures and Routines
  - Using Lists
  - Node Types

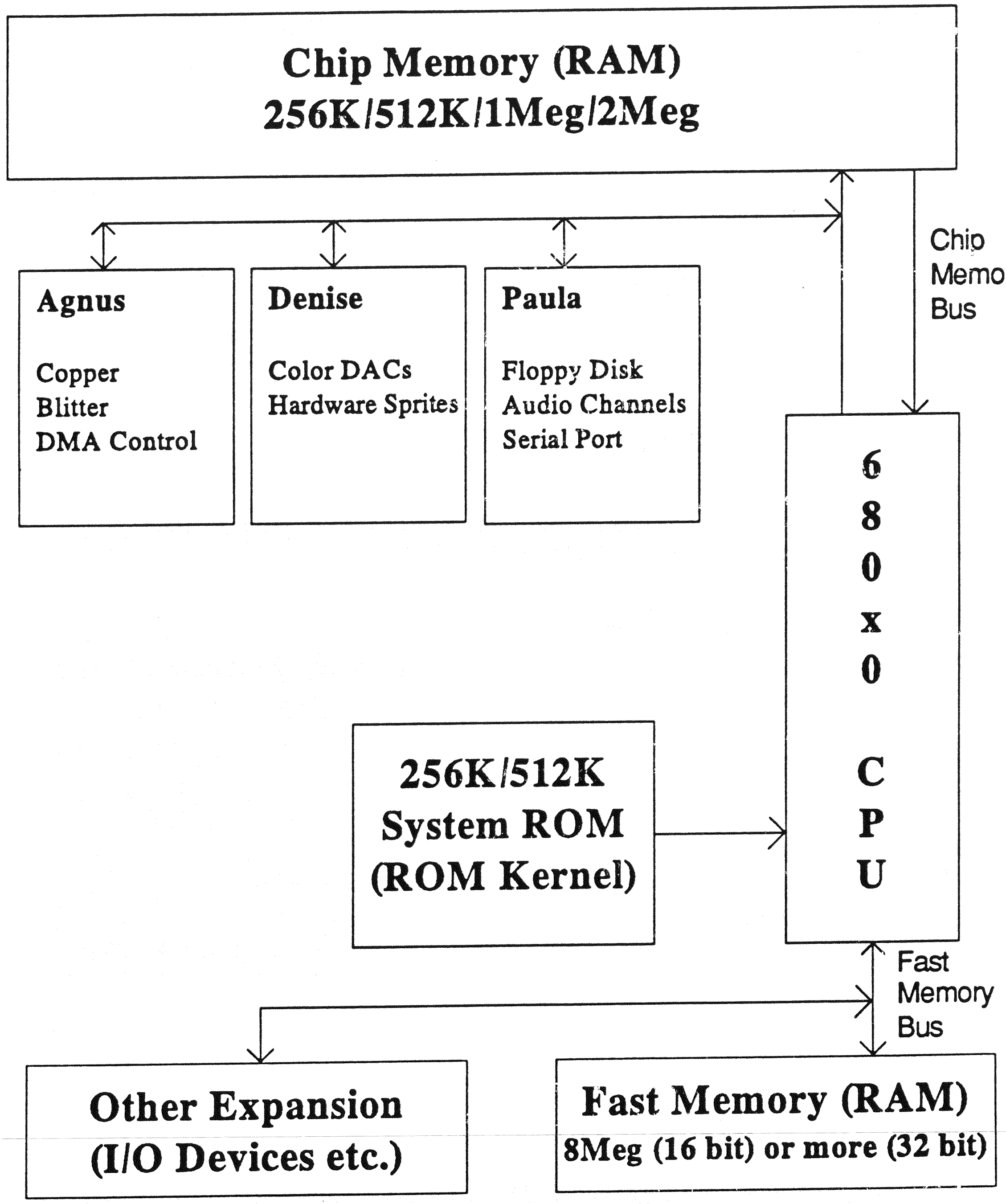
Advanced EXEC Concepts

- Memory Management
  - Routines
  - Using AllocMem
  - System Memory Organization
- Tasks
  - Data Structures and Routines
  - Launching a Task
  - Controlling Task Scheduling
- Signals and Signal Masks
- Events and Wait()
- Message Ports/InterProcess Communication
- EXEC I/O
  - IORequests
  - Device Example - Using The timer.device
- Guru Meditations and System Errors

Tips and Caveats

References

Simplified Amiga Hardware Block Diagram


Chip Memory (RAM)
256K/512K/1Meg/2Meg
Chip Memory Bus
Agnus
Copper
Blitter
DMA Control
Denise
Color DACs
Hardware Sprites
Paula
Floppy Disk
Audio Channels
Serial Port
680x0 CPU
256K/512K System ROM (ROM Kernel)
Fast Memory Bus
Other Expansion (I/O Devices etc.)
Fast Memory (RAM)
8Meg (16 bit) or more (32 bit)

Hardware In Detail

Agnus - Copper/CoProcessor

The Copper has 3 instructions:

Move - Move a value into a custom chip register.

Wait - Wait for the video beam to reach a specific position (x,y).

Skip - Skip the next instruction if the video beam has already reached a specific position (x,y).

Denise - Color DACs (Digital To Analog Converters)

- The registers on Denise determine the color of the display and drive the display itself. Denise is also responsible for control of the hardware sprites.

Paula - Audio Channels and I/O

- Paula controls the stereo audio channels and plays a major role in floppy disk control as well as joystick/mouse, serial and parallel port control which it shares with the 8520 CIAs.

Chip Memory/Chip Bus

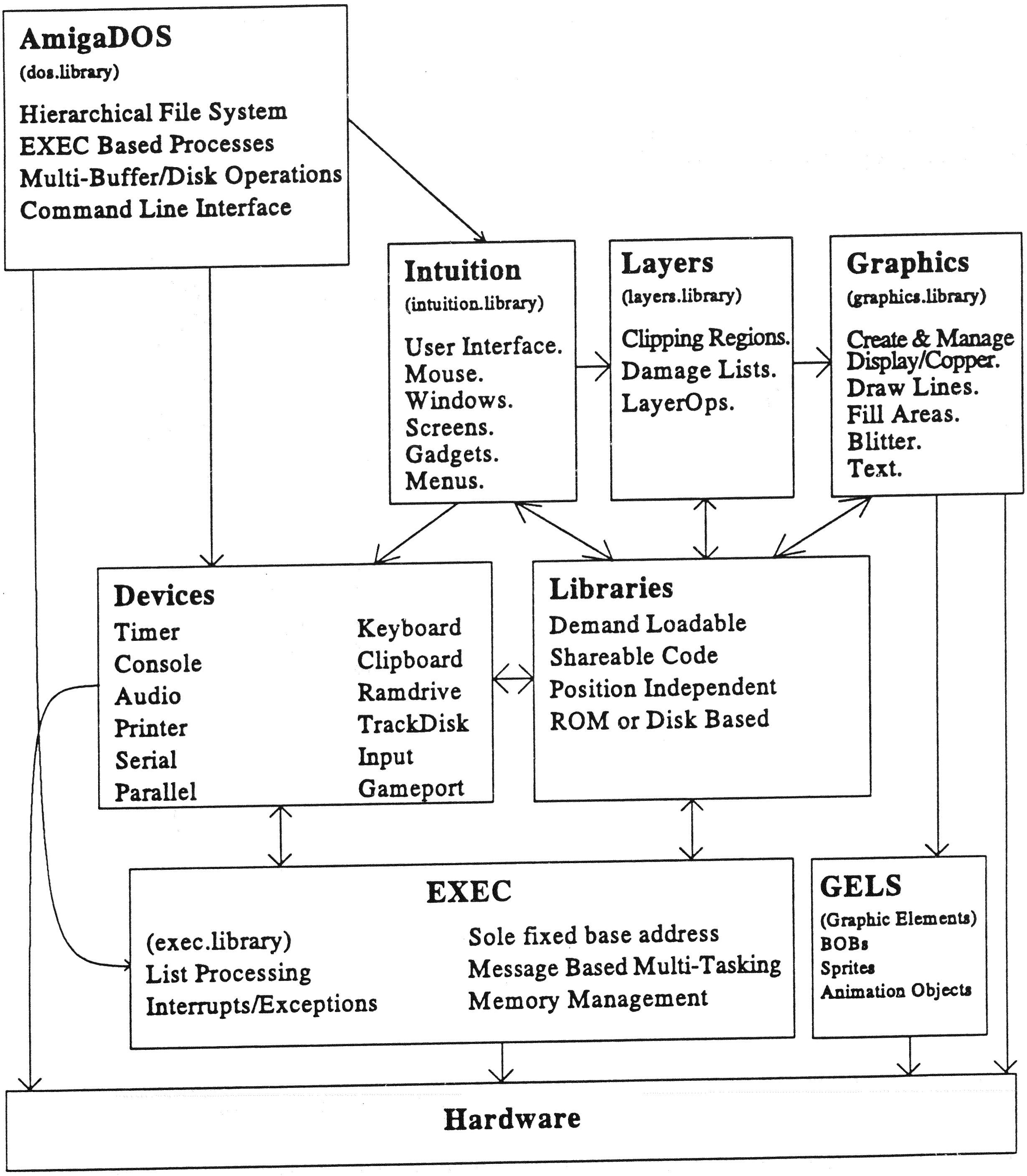
- All the custom chips and a limited addressable range of memory live on this data bus. This bus is also shared with the 68000 processor.

Fast Memory/Expansion Bus

- All add-on memory (above and beyond chip-memory) and devices (e.g. disk controllers, frame buffers) are placed on this bus. The 68000 (or other main processor) accesses this bus without contention (i.e. it runs at full speed at all times).
- Devices on the Expansion bus may DMA to each other without interfering with devices on the Chip Bus.
- Expansion devices are mapped above the 8 Meg RAM expansion space. Any addressable memory (e.g. buffer memory) on these devices is configured in in 64K chunks.

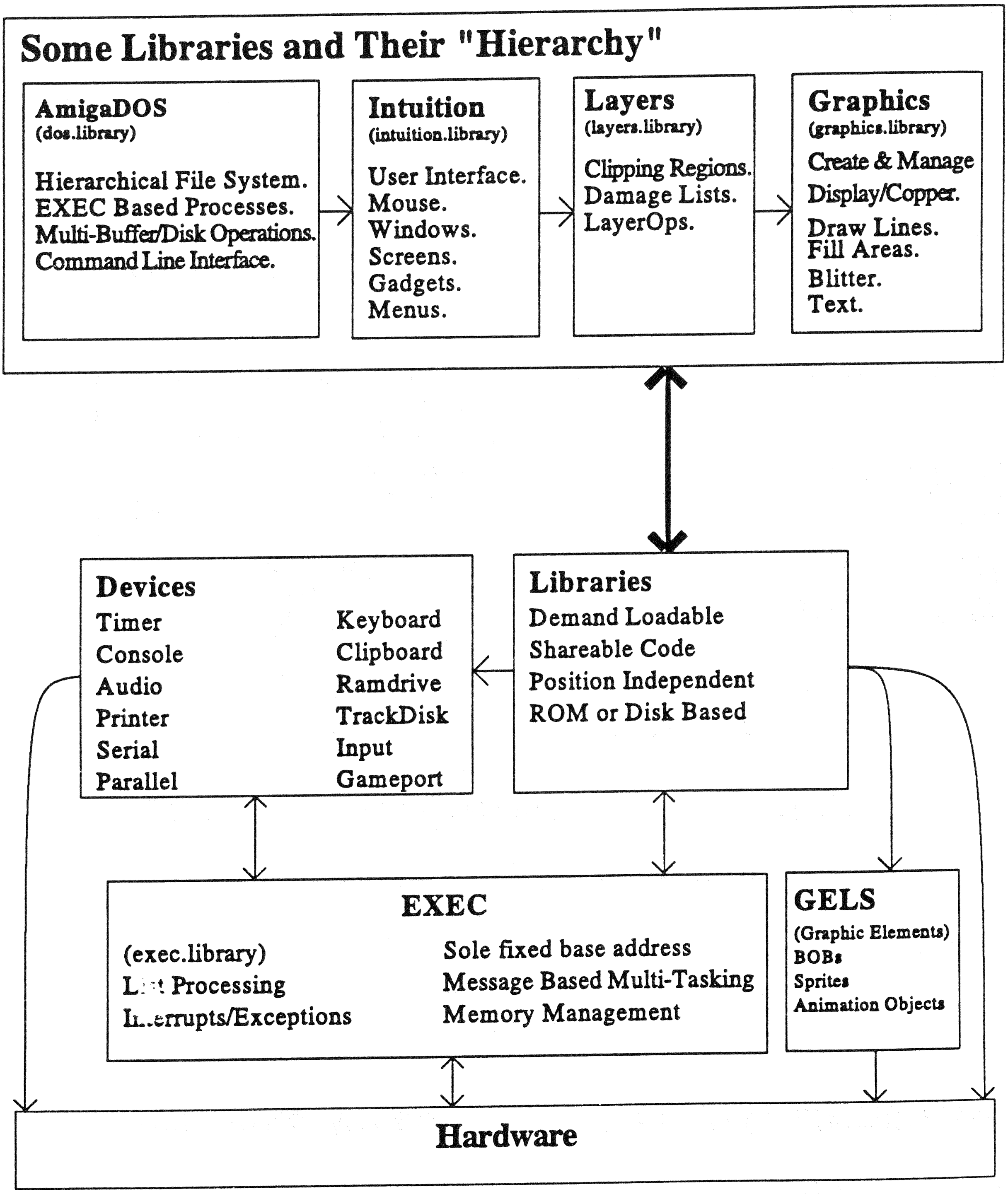
ROM Kernel

- This 256K (1.3) or 512K (2.0) of ROM (Kickstart Writable Control Store on A1000's) contains the instructions that make up the Amiga system software. Without these routines the Amiga would be a useless machine


AmigaDOS
(dos.library)
Hierarchical File System
EXEC Based Processes
Multi-Buffer/Disk Operations
Command Line Interface
Intuition
(intuition.library)
User Interface.
Mouse.
Windows.
Screens.
Gadgets.
Menus.
Layers
(layers.library)
Clipping Regions.
Damage Lists.
LayerOps.
Graphics
(graphics.library)
Create & Manage
Display/Copper.
Draw Lines.
Fill Areas.
Blitter.
Text.
Devices
Timer
Console
Audio
Printer
Serial
Parallel
Keyboard
Clipboard
Ramdrive
TrackDisk
Input
Gameport
Libraries
Demand Loadable
Shareable Code
Position Independent
ROM or Disk Based
EXEC
(exec.library)
List Processing
Interrupts/Exceptions
Sole fixed base address
Message Based Multi-Tasking
Memory Management
GELS
(Graphic Elements)
BOBs
Sprites
Animation Objects
Hardware

Amiga ROM Kernel Block Diagram

Amiga ROM Kernel Block Diagram



Some Libraries and Their "Hierarchy"

AmigaDOS
(dos.library)

Hierarchical File System.
EXEC Based Processes.
Multi-Buffer/Disk Operations.
Command Line Interface.

Intuition
(intuition.library)

User Interface.
Mouse.
Windows.
Screens.
Gadgets.
Menus.

Layers
(layers.library)

Clipping Regions.
Damage Lists.
LayerOps.

Graphics
(graphics.library)

Create & Manage
Display/Copper.
Draw Lines.
Fill Areas.
Blitter.
Text.

Devices

Timer
Console
Audio
Printer
Serial
Parallel
Keyboard
Clipboard
Ramdrive
TrackDisk
Input
Gameport

Libraries

Demand Loadable
Shareable Code
Position Independent
ROM or Disk Based

EXEC

(exec.library)
List Processing
Interrupts/Exceptions
Sole fixed base address
Message Based Multi-Tasking
Memory Management

GELS

(Graphic Elements)
BOBs
Sprites
Animation Objects

Hardware

Just What Is A Library Anyway?

A Library is defined as a collection of related routines.

A Library can be either ROM or Disk Based (e.g. graphics.library is in ROM, info.library is on Disk).

A routine in a Library is accessed by jumping to a defined offset from the base address of the Library.

A Library is generally re-entrant - it's code can be shared and executed by more than one task at a time.

A Library is position independent - it can be loaded at an arbitrary location in memory.

A Library is opened using the EXEC routine OpenLibrary and closed using CloseLibrary.

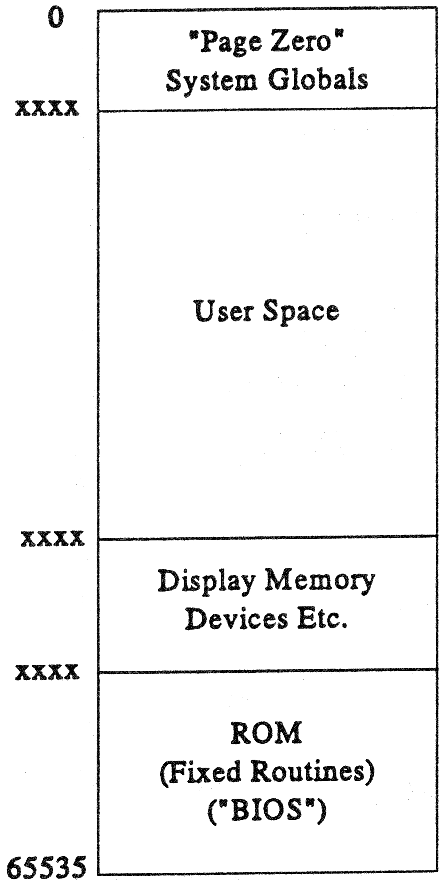
OpenLibrary increments the "usage" count of the Library and CloseLibrary decrements this counter.

When the "usage" or "open" count of a Library reaches 0, EXEC is permitted to flush the Library from memory if the space occupied by the Library is needed by the system and the Library is RAM resident (e.g. loaded from Disk).

Multiple versions of a Library can be distinguished by a version number (1.2 = V33, 1.3 = V34). Applications should open the lowest version that meets their needs.

Comparison Of Typical PC Memory Map vs Amiga Memory Layout

Typical MicroComputer Memory Map
(Apple II/Commodore 64/IBM PC)



0

"Page Zero"
System Globals

xxxx

User Space

xxxx

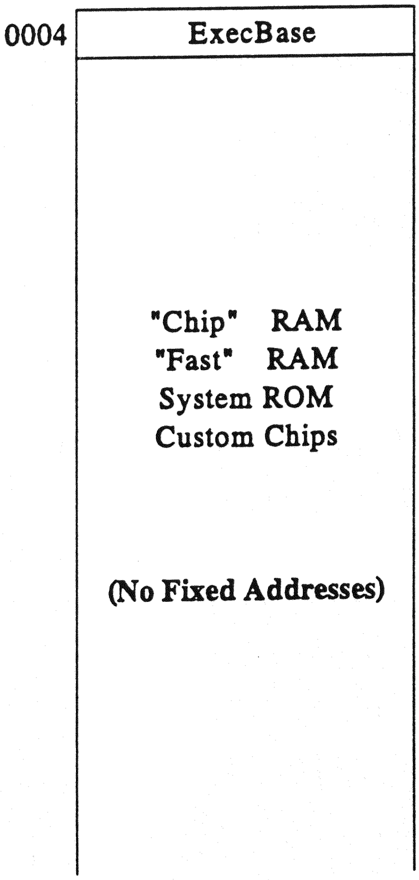
Display Memory
Devices Etc.

xxxx

ROM
(Fixed Routines)
("BIOS")

65535

Commodore-Amiga Memory Map



0004

ExecBase

"Chip" RAM
"Fast" RAM
System ROM
Custom Chips

(No Fixed Addresses)

Gaining Access To The Amiga ROMs

```
_AbsExecBase       EQU    4       ; normally XREF this
_LVOOpenLibrary    EQU    -552    ; normally XREF this

_IntuitionBase:    dc.l   $0000
_ILibName:         dc.b   'intuition.library',0

moveq.l    #33,d0                   ; want V33 or greater
lea        _ILibName,a1             ; get library name
movea.l    _AbsExecBase,a6          ; library base ptr
jsr        _LVOOpenLibrary(a6)      ; call OpenLibrary
move.l     d0,_IntuitionBase        ; store the result
beq.s      my_error_abort           ; check result
     .
     .
     .
```

This moves the base address (memory location) of ExecBase into register a6 and then calls the EXEC routine OpenLibrary, which is an offset into the exec.library.

Higher level languages do this dirty work for you.

LVO == *Library Vector Offset*

Using The Autodocs

```
exec.library/OpenLibrary

NAME
  OpenLibrary -- gain access to a library (optional version)

SYNOPSIS
  library = OpenLibrary(libName, version)
  D0                    A1       D0

FUNCTION
  This function returns a pointer to a library that was
  previously installed into the system.  If the requested
  library is exists, and if the library version is greater
  than or equal to the requested version, then the open
  will succeed.

INPUTS
  libName - the name of the library to open

  version - the version of the library required.

RESULTS
  library - a library pointer for a successful open, else
  zero

SEE ALSO
  CloseLibrary
```

The Autodocs can be an invaluable reference for the latest up to date calling conventions for Amiga routines. Full information on calling conventions for C as well as Assembly and cross reference to related routines is included for every entry.

New functions generally have a version number in parentheses at the end of the name line, e.g: (V36).

Using OpenLibrary()/CloseLibrary()

```
#include <exec/exec.h>                /* Include all the Exec Headers */
#include <intuition/intuition.h>     /* Include all the Intuition Structures */
#include <dos/dos.h>

extern struct Library    *OpenLibrary();
struct Library           *IntuitionBase = NULL;

main()
{
   if ((IntuitionBase = OpenLibrary("intuition.library", 33L)) == NULL) {
       exit(RETURN_FAIL);
   }

   /* We can now call any Intuition routine */

   if (IntuitionBase) CloseLibrary(IntuitionBase);
}
```

The include files are provided by Commodore-Amiga and are included with all commercial compilers. Their organization follows the logical hierarchy of the hardware and ROM Kernel. Be sure your include files are current with the target version of the operating system you are running.

OpenLibrary() returns the base address of intuition.library if it succeeds. When we exit, we check to see if IntuitionBase does indeed point to a valid library base and we CloseLibrary Intuition if it does.

When we call an Intuition routine, the compiler is smart enough to load the base address of the Intuition Library as reflected in our copy of "IntuitionBase" and call the proper offset into the Library for the routine in question.

Libraries in Detail

When opening a library, you must use the following naming conventions for the library base pointer:

BaseName	Library	Include File
ROM Libraries as of 1.3:		
DiskfontBase	diskfont.library	libraries/diskfont.h
DOSBase	dos.library	libraries/dos.h
ExecBase	exec.library	exec/execbase.h
ExpansionBase	expansion.library	libraries/expansion.h
GfxBase	graphics.library	graphics/gfxbase.h
IntuitionBase	intuition.library	intuition/intuition.h
LayersBase	layers.library	graphics/layers.h, clip.h
MathBase	mathffp.library	libraries/mathffp.h
MathTransBase	mathtrans.library	libraries/mathffp.h
RomBootBase	romboot.library	libraries/romboot_base.h
ROM Libraries introduced in 2.0:		
GadToolsBase	gadtools.library	libraries/gadtools.h
KeymapBase	keymap.library	devices/keymap.h
UtilityBase	utility.library	utility/#?.h
WorkbenchBase	workbench.library	workbench/#?.h
Disk Based Libraries as of 1.3:		
IconBase	icon.library	workbench/icon.h
	info.library	not documented
MathIeeeDoubBasBase	mathieeedoubbas.library	libraries/mathlibrary.h libraries/mathieedp.h
MathIeeeDoubTransBase	mathieeedoubtrans.library	libraries/mathlibrary.h libraries/mathieeedp.h
TranslatorBase	translator.library	libraries/translator.h
	version.library	not documented
Disk Based Libraries Introduced in 2.0:		
AslBase	asl.library	libraries/asl.h and aslbase.h
CxBase	commododities.library	libraries/commodities.h
IFFParseBase	iffparse.library	libraries/iffparse.h
RexxSysBase	rexxsyslib.library	rexx/rxslib.h

The general rule of thumb is to close libraries in the reverse order from which you opened them.

EXEC Lists

Lists are a fundamental data structure in Computer Science.

Lists are the core data structure of EXEC.

An understanding of Lists is necessary to write programs that deal with all other aspects of EXEC.

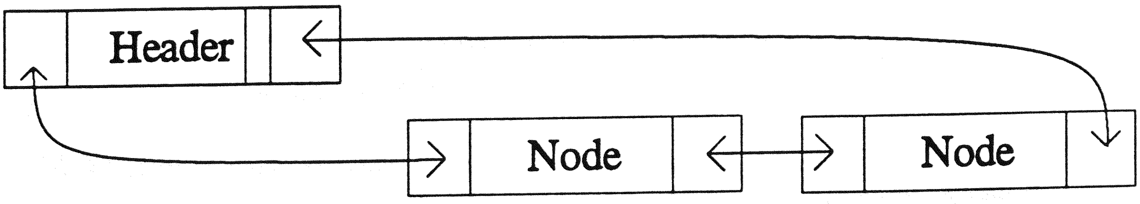
EXEC Lists are extremely efficient.

Your own applications can easily use and benefit from EXEC Lists.

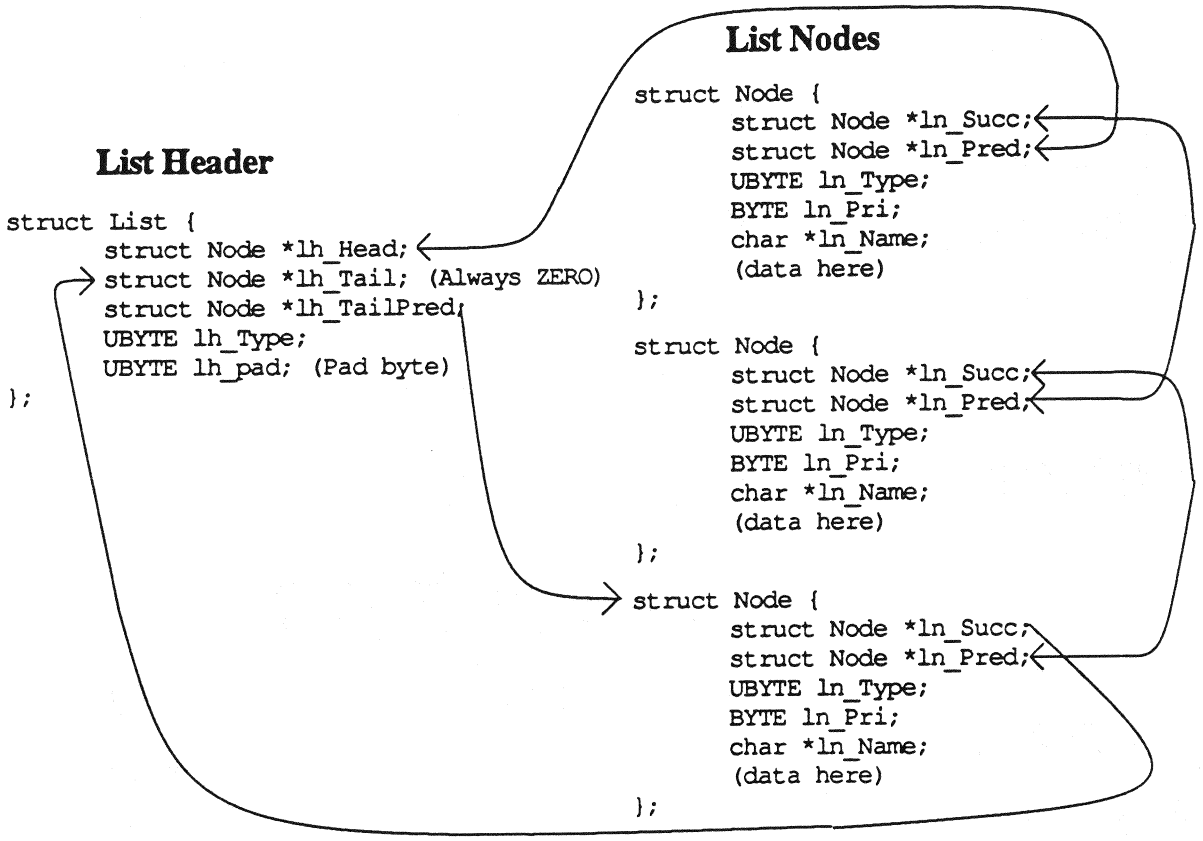
Some ROM Kernel subsystems that directly or indirectly use Lists (Nodes):

- Memory Management
- Tasks
- Interrupts
- Messages/MessagePorts
- Device I/O Requests

Lists can be sorted by name or priority.



EXEC Lists In Detail



List Header

```
struct List {
        struct Node *lh_Head;
        struct Node *lh_Tail; (Always ZERO)
        struct Node *lh_TailPred;
        UBYTE lh_Type;
        UBYTE lh_pad; (Pad byte)
};
```

List Nodes

```
struct Node {
        struct Node *ln_Succ;
        struct Node *ln_Pred;
        UBYTE ln_Type;
        BYTE ln_Pri;
        char *ln_Name;
        (data here)
};

struct Node {
        struct Node *ln_Succ;
        struct Node *ln_Pred;
        UBYTE ln_Type;
        BYTE ln_Pri;
        char *ln_Name;
        (data here)
};

struct Node {
        struct Node *ln_Succ;
        struct Node *ln_Pred;
        UBYTE ln_Type;
        BYTE ln_Pri;
        char *ln_Name;
        (data here)
};
```

NewList() - pass this a pointer to a List structure to initialize a List
Insert() - inserts a node into a list after a specified node already in the list
Remove() - removes a specified node
AddHead() - insert a node at the head of a list; more efficient than Insert() for Head
RemHead() - remove node from the head of a list
AddTail() - insert a node at the tail of a list; more efficient than Insert() for Tail
RemTail() - remove node from the tail of a list
Enqueue() - insert a node into a list in priority order
FindName() - search a given list for a named node

Note that the Head/Tail is combined. It removes the need for a special case test for the empty list.

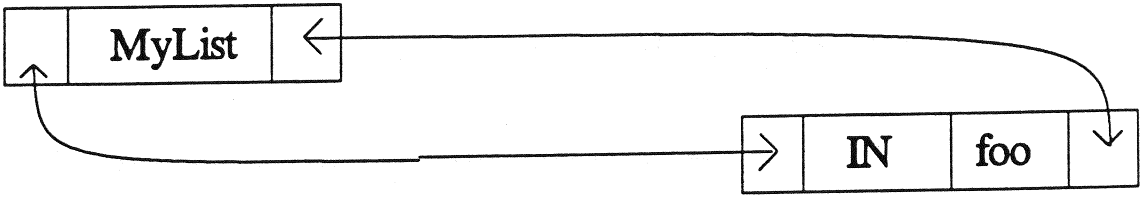
You should also think about using MinLists/MinNodes - all these routines except Enqueue() will work with them.

Using Lists

```
#include <exec/types.h>
#include <exec/nodes.h>
#include <exec/lists.h>
#include <exec/memory.h>

extern void        *AllocMem();
extern struct Node *RemHead(), *FindName();

struct MinList     *MyList = NULL;

struct MyNode {
      struct Node INode;      /* This is a Node */
      long        foo;        /* Plus some useful data */
} *IN = NULL;

/* Free the memory used by a MyNode passed in 'IN' */
FreeNode(IN)
register struct MyNode *IN;
{
      if (IN) {
            FreeMem(IN, (long) sizeof(struct MyNode));
      }
}

/* Free all MyNodes in a given MinList passed in 'list'*/
FreeAll(list)
register struct MinList *list;
{
      register struct MyNode *IN;

      while (IN = (struct MyNode *) RemHead(list)) {
            FreeNode(IN);
      }
}

/* Free a given MinList passed in 'list' */
FreeList(list)
register struct MinList *list;
{
      if (list) {
            if (list->mlh_Head) FreeAll(list);
            FreeMem(list, (long) sizeof(struct MinList));
      }
}
```

Using Lists (contd.)

```
/*
* Our simple example will allocate memory for a MinList and a
* custom Node structure. It will then initialize the list using
* NewList(), stuff some data into the Node structure and add it
* to the end of the MinList using AddTail(). It then checks to
* see that it did indeed succeed by doing a FindName() on the
* Node in the MinList. We finish up by freeing *all* the elements
* (or Nodes) in the MinList and the MinList itself.
*/
main()
{
        struct MyNode *sNode;

        if (!(MyList = AllocMem((long) sizeof(struct MinList),
                                MEMF_FAST | MEMF_CLEAR))) {
                printf("couldn't allocate MinList");
                exit(1);
        }
        NewList(MyList);

        if (!(IN = AllocMem((long) sizeof(struct MyNode),
                                MEMF_FAST | MEMF_CLEAR))) {
                printf("couldn't allocate MyNode");
                goto die;
        }

        IN->foo = 5L;
        IN->INode.ln_Name = (char *) "our node";
        AddTail(MyList, IN);

        if (sNode = (struct MyNode *) FindName(MyList, "our node")) {
                printf("found our node");
        }

die:    FreeList(MyList);
}
```



Note that by definition we cannot use Enqueue() with MinLists/Nodes

Nodes

EXEC understands some special Node Types (defined in exec/nodes.h):

NT_UNKNOWN	- Set for foreign node types
NT_TASK	- This node is a Task
NT_INTERRUPT	- This node is an Interrupt
NT_DEVICE	- This node is a Device driver
NT_MSGPORT	- This node is a MsgPort
NT_MESSAGE	- This node is a Message
NT_FREEMSG	- This node is a Message available for re-use
NT_REPLYMSG	- This node is a Reply Message
NT_RESOURCE	- This node is a Resource
NT_LIBRARY	- This node is an Exec Library
NT_MEMORY	- This node is in the Memory pool
NT_SOFTINT	- This node is a Software Interrupt
NT_FONT	- This node is a Font
NT_PROCESS	- This node is a DOS Process
NT_SEMAPHORE	- This node is a Semaphore
NT_SIGNALSEM	- This node is a Signal Semaphore
NT_BOOTNODE	- This node is an AutoBoot device

EXEC Memory Management

Remember, there are two types of memory on the Amiga. They are Chip Memory, and Fast Memory.

AllocMem()
This routine is the workhorse of EXEC. This routine is used by both the system and applications to allocate individual regions of memory.

FreeMem()
The converse to AllocMem(), this routine will return memory that was previously allocated back to the system. Be careful that you free only memory which was previously allocated. Under V1.3 and earlier systems, freeing memory not allocated will crash the system with a "Free Twice" Guru Alert (81000009).

AllocEntry()
Will allocate multiple regions of memory defined using MemLists. These regions can then be managed using Allocate()/Dealllocate().

FreeEntry()
The converse to AllocEntry.

Allocate()
Will allocate memory from a given freelist (MemList).

Deallocate()
The converse to Allocate().

AvailMem()
Given a memory type (MEMF_CHIP or MEMF_FAST), this call will return the amount of free memory of that type in bytes remaining in the system.

TypeOfMem()
This routine, given a valid RAM address will return the type of memory referenced by that address (MEMF_CHIP or MEMF_FAST

AllocAbs()
This routine will attempt to allocate a range of memory beginning at a user specified base address.

Using AllocMem()

```
#include <proto/exec.h>
#include <exec/memory.h>

APTR mydata = NULL;

main()
{
      mydata = AllocMem(128L, MEMF_CHIP | MEMF_CLEAR);
      if (mydata) FreeMem(mydata, 128L);
}
```

Any combination of the following flags (except MEMF_CHIP | MEMF_FAST) may be used to specify the characteristics of the memory to be allocated:

MEMF_PUBLIC: Memory must not be mapped, swapped, or otherwise made non-addressable. ALL MEMORY THAT IS REFERENCED VIA INTERRUPTS AND/OR BY OTHER TASKS MUST BE EITHER PUBLIC OR LOCKED INTO MEMORY! This includes both code and data (e.g. Message Ports). This flag will become significant if the Amiga hardware moves to a Memory Management Unit and/or the software moves to a virtual memory system(or a new flag may be used).

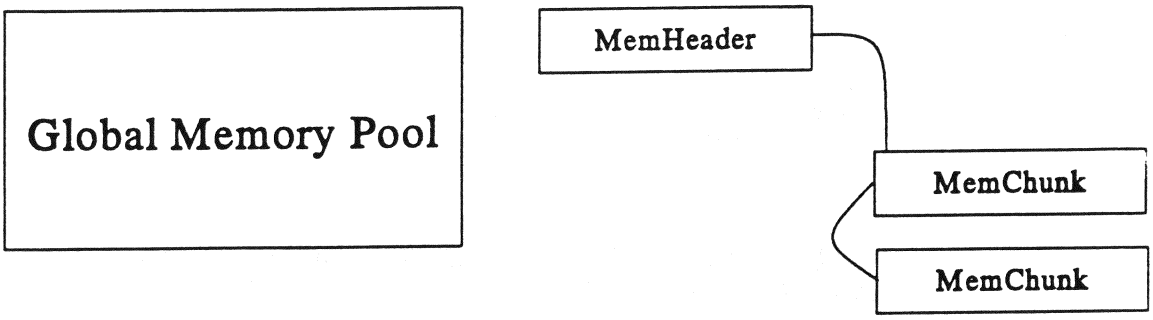
MEMF_CHIP: Only certain parts of memory are reachable by the special chip sets' DMA circuitry. Anything that will use on-chip DMA must be in memory with this attribute. DMA includes screen memory, things that are blitted, audio blocks, raw disc buffers, etc.

MEMF_FAST: Non-chip (exdpansion) memory. It is possible for the processor to get locked out of chip memory under certain conditions. FAST memory does not suffer from this problem.Since not all machines are guaranteed to have FAST memory, do not insist on allocating MEMF_FAST.

MEMF_CLEAR: The memory will be initialized to all zeros.

AllocMem will always allocate FAST memory if it is available.

EXEC Memory Organization



Global Memory Pool

MemHeader

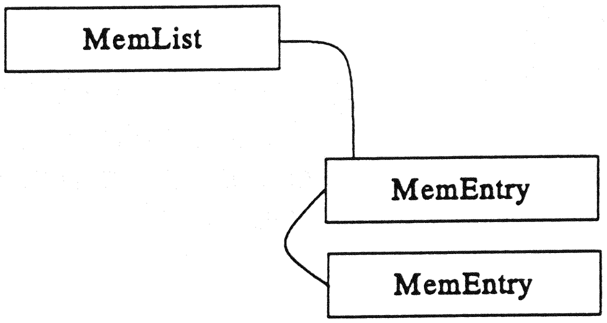
MemChunk

MemChunk

Global Memory Organization

A MemChunk is simply a region of memory (# of bytes)

The MemHeader specifies characteristics (Chip/Fast etc.), bounds and total free bytes of all the Chunks.



MemList

MemEntry

MemEntry

Allocating Multiple Memory Blocks Using AllocEntry

WARNING: read the AllocEntry Autodoc. AllocEntry returns a special non-zero value on failure!

Tasks

A Task is a thread of execution.

Individual Tasks each have their own processor state or context.

Tasks are heavily based on EXEC Lists; each Task begins with a Node.

EXEC completely manages the scheduling of Tasks on the following basis:

> The highest priority Task is run and continues executing until either:
>
> 1) a higher priority Task becomes active.
> 2) the running Task exceeds it's Quantum (~50ms).
> 3) the running Task needs to wait for some external event to occur before it can continue.

The Task that is running is said to be "Running" or on the Run Queue (there is currently only one Task on the Run Queue at any given time).

A Task that is in line to be executed is said to be "Ready" or on the Ready Queue.

A Task that is waiting for an external event is said to be "Waiting" or on the Wait Queue.

When a Task that is Waiting receives one of it's external events, or Signals, it is placed on the Ready Queue.

The Ready Queue is kept in priority-sorted order.

Tasks cannot make calls to DOS; you must use a Process to make calls to DOS.

This method of scheduling is known as "Round-Robin Pre-Emptive Task Scheduling"

Tasks In Detail

AddTask() - add a Task to the system given an initial and final code address. The Task must have space for it's local stack allocated (200 bytes min recommended for EXEC only, 4K min recommended for dealing with DOS - but your Task will be part of a Process structure).

RemTask() - remove a Task with a given name. RemTask(NULL) will remove one's self. Be sure you've freed all allocated resources first!

FindTask() - find a Task with a given name. FindTask(NULL) will find the current Task.

AllocSignal() - allocate a signal bit from the Tasks current pool (32 signals per Task or one longword, 16 usable by you).

FreeSignal() - free a previously allocated signal so it may be re-used.

AllocTrap() - allocate a processor trap vector.

FreeTrap() - free a trap vector for re-use.

Forbid() - prohibits all Task rescheduling.

Permit() - re-enables Task rescheduling.

Disable() - prohibit all Interrupt processing and Task rescheduling.

Enable() - re-enable Interrupt processing and Task rescheduling.

Wait() - relinquish the CPU and wait for one or more Signals.

CreateTask() - initialize and launch a new Task (compiler library routine).

DeleteTask() - remove a running Task (compiler library routine - simply calls RemTask()).

The Task Structure

```
extern struct Task {
       struct Node tc_Node;         /* To link to the next task */
       UBYTE tc_Flags;              /* Task Flags */
       UBYTE tc_State;              /* Task state */
       BYTE tc_IDNestCnt;           /* Interrupt disabled nest count */
       BYTE tc_TDNestCnt;           /* Task disabled nest count (Forbid()) */
       ULONG tc_SigAlloc;           /* Signals allocated */
       ULONG tc_SigWait;            /* Signals we're waiting for */
       ULONG tc_SigRecvd;           /* Signals we've received */
       ULONG tc_SigExcept;          /* Signals we will take exceptions for */
       UWORD tc_TrapAlloc;          /* Traps allocated */
       UWORD tc_TrapAble;           /* Traps enabled */
       APTR tc_ExceptData;          /* points to exception data */
       APTR tc_ExceptCode;          /* points to exception code */
       APTR tc_TrapData;            /* points to trap data */
       APTR tc_TrapCode;            /* points to trap code */
       APTR tc_SPReg;               /* Stack pointer */
       APTR tc_SPLower;             /* Stack lower bound */
       APTR tc_SPUpper;             /* Stack upper bound */
       VOID (*tc_Switch)();         /* See below */
       VOID (*tc_Launch)();         /* See Below */
       struct List tc_MemEntry;     /* allocated memory */
       APTR tc_UserData;            /* per-task user data */
};
```

The tc_Switch and tc_Launch vectors point to the code that is executed when a task is removed from the run queue (pre-empted) and when a task is put on the run queue (started).

Usually these routines save all the processor registers on your stack during Switch and restore them upon Launch. It is possible to wedge into these routines to increase their functionality. For instance, if you have a peripheral 68881 under 1.3 or greater, a vendor may supply the necessary routines to save and restore the 68881 registers to be wedged into the Switch and Launch vectors.

Launching A Task

```
#include <exec/exec.h>
#include <proto/exec.h>

struct Task          *theTask = NULL;
char                 *myTaskName = "myTask";
ULONG                myvar;

/* The SubTask waits on a signal from the main program */
void myTask()
{
#ifdef AZTEC_C
        geta4();
#endif

        myvar = 100;
        (void) Wait(0L);            /* Wait forever */
}

void main(int argc, char **argv)
{
        myvar =0;
        theTask = CreateTask(myTaskName, 0L, myTask, 4000L);

        /* Must synchronize with SubTask first! This does not guarantee it!*/
        Delay(5L);
        printf("myvar is: %ld\n", myvar);

        /* We know it's safe to remove theTask */
        if (theTask) DeleteTask(theTask);
}
```

Note that CreateTask does the dirty work necessary for calling AddTask(), and DeleteTask() calls RemTask(). CreateTask() also handles all the necessary memory and stack allocation.

The subtask shares the same data space with the parent program.

The subtask may be safely removed if it is not going to be awakened and placed on the ready/run queue (it is waiting for a signal/condition which will not occur).

Enable/Disable - Forbid/Permit

These routines should be used with extreme care. Incorrect use or overuse of these routines can seriously degrade the preformance of the Amiga multitasking environment. When should you use these functions?

Forbid() is sometimes used when accessing system data structures which are dynamic in nature and which other tasks may access/modify. It will ensure that your task can examine these structures in a consistent fashion. Another mechanism called Semaphores may also be used for this type of activity, and is often preferred. See the Intuition routine LockIBase() for a case where Forbid()'ing is not kosher.

Be aware that calling Wait() or calling any routine which may Wait() will effectively relinquish the processor to other tasks. When the Task is re-scheduled, you will re-enter the Forbid() state. (Note that any file I/O and printf() cause a Wait()).

Disable() is similiar to Forbid(), but also prevents all interrupts from occuring (such as an interrupt from a Disk controller). You might Disable() if the data structure you are acessing is shared by Interrupt code. Since so much of the normal activity of the system (e.g. updating the display, controlling the mouse) relies on near real-time interrupts, you should never Disable() for more than an instant. Also, do not attempt to call routines that reqiure a multitasking environment while Disable()'ed. You will hang the system (e.g. don't call printf() or attempt to do other I/O).

Both Forbid() and Disable() may be nested.

It is never necessary to both Disable() and Forbid(). Since Disable() prevents interrupts, it also prevents task scheduling.

Signals and Signal Masks

Every task is reserved a ULONG field which identifies 32 possible signals which it can receive. These signals represent some event in the EXEC which may be caused by a software or hardware interrupt. Of these 32 signals, only the upper 16 are usable by your application. The rest are reserved for system use.

So, every bit in the ULONG field uniquely identifies every possible signal for a given task. Signals (bits) from this field are reserved (allocated) with the EXEC system call AllocSignal() and made available for re-use with FreeSignal().

AllocSignal(-1L) will allocate the next available signal bit. If none is available, it will return -1. In the case below, AllocSignal returns the signal number which was allocated, or 17 (range 0..31).

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

31 Before AllocSignal(-1L) 16

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

After AllocSignal(-1L)

To re-create exactly which bit was set from this number, we must create a Signal Mask. This is done by taking the longword 1 and shifting it left by the signal number:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

"1L"

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

"1L << 17"

We will need to create masks like this in order to create combinations of signal bits to pass to Wait()

Events and Wait()

Signals are used to wake up tasks upon the occurrence of some external event. This is done using the EXEC system call Wait(). This call accepts a ULONG parameter in which all the signal bits for events to wake up the task are set. So, if we want to be woken up by one event whose signal mask is "signal1", we call Wait() like this:

```
Wait(signal1);
```

If we wish to be woken up by one or more events, we OR their signal masks together and pass them to Wait():

```
Wait(signal1 | signal2 | ... | signaln);
```

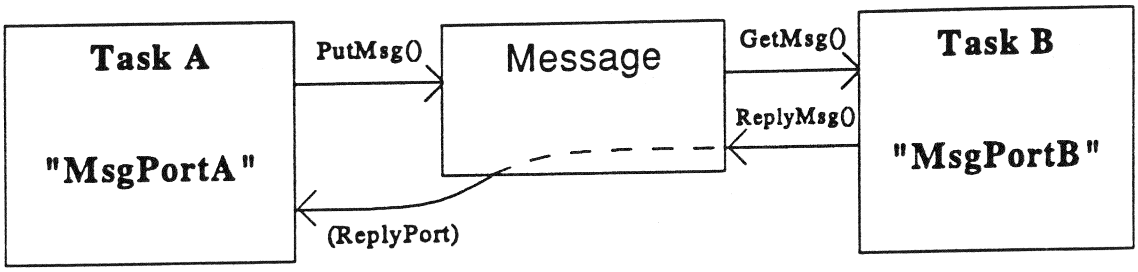
Wait() actually returns a ULONG value which represents which signals actually woke the task up. It is the applications' responsibility to determine which signals woke it up and what action to perform.

```
sigsrecvd = Wait(signal1 | signal2 | ... | signaln);
if (sigsrecvd & signal1) { /* do stuff for signal1 }
if (sigsrecvd & signal2) { /* do stuff for signal2 }
if (sigsrecvd & signaln) { /* do stuff for signaln }
```

Wait() will clear any signals which may have been set in the task block by EXEC. Calling Wait() a second time will put the task to sleep untill the next set of events occur.

When Wait() is called, the task gets placed on the Wait Queue untill EXEC determines that an event has occured which will wake that task up (based on the signal masks passed to Wait()). When the task is woken up, the tc_SigsRecvd field in the task block gets set by EXEC and EXEC places the task on the Ready Queue.

Message Ports/IPC (InterProcess Communication)



Task A creates it's own uniquely named MsgPort "MsgPortA".

Task B creates it's own uniquely named MsgPort "MsgPortB", then Wait()'s on it's Signal Bit.

Task A calls FindPort("MsgPortB").

Task A builds a Message structure, fills it with some data and does a PutMsg() to the port returned from FindPort("MsgPortB")

Task A is now free to continue with other processing or to block for a reply from Task B by Wait()'ing on the Signal Bits from the Message Reply Port.

Task B, which has been Wait()'ing on the Signal Bits for MsgPortB, is woken up by this event.

Task B does a GetMsg() to read the message and remove it from the port

When Task B is finished processing the data in the message, it replies to Task A using ReplyMsg(). This tells Task A know that it can re-use the memory from the Message.

The analogy used here is that of an answering machine.

MsgPorts

The following function is available in most compiler libraries already. It will allocate a message port with the appropriate signal bits so that it may be used for communication with other processes.

Be sure to use DeletePort() before exiting your application!

Be sure to reply to all outstanding messages before calling DeletePort() (using ReplyMsg()).

```
struct MsgPort *CreatePort(name, pri)
char *name;
BYTE pri;
{
      UBYTE  sigBit;
      struct MsgPort *port;

      if ((sigBit = AllocSignal(-1L)) == -1)
         return ((struct MsgPort *) NULL);

      port = AllocMem((long) sizeof(*port), MEMF_CLEAR |  MEMF_PUBLIC);
      if (port == 0) {
         FreeSignal(sigBit);
         return ((struct MsgPort *) NULL);
      }

      port->mp_Node.ln_Name = name;
      port->mp_Node.ln_Pri = pri;
      port->mp_Node.ln_Type = NT_MSGPORT;

      port->mp_Flags = PA_SIGNAL;
      port->mp_SigBit = sigBit;
      port->mp_SigTask = FindTask(0L);

      if (name != 0)
         AddPort(port);
      else
         NewList(&port->mp_MsgList);

      return (port);
}

DeletePort(port)
struct MsgPort  *port;
{
      FreeSignal(port->mp_SigBit);
      RemPort(port);
      FreeMem(port, (long) sizeof(*port));
}
```

Messages

```
struct Message {
      struct Node mn_Node;
      struct MsgPort *mn_ReplyPort;/* Port where replymsg will go to */
      UWORD mn_Length;              /* Length of Message body in Bytes */
      (Data here)
}
```

A Message can contain any data you desire to place in one.

Messages live in public memory (MEMF_PUBLIC), they are sent from Task to Task by passing a pointer to their location in memory. This is much faster than copying the whole message body.

Do not re-use a Message until it has been replied to by the recepient.

When you receive a Message, copy it's data to your own local buffers as soon as possible, then reply to it immediately using ReplyMsg().

How To Receive and Act On Messages

Many Messages may queue up at a port at one time, but you will receive only one signal for all of them. You should process all available messages and then Wait for more to come in.

```
for (;;) {  /* Loop Forever or until break; */
     if ((msg = GetMsg(Port)) == NULL) {
        (void) Wait(SignalMask);
        continue; /* Try GetMsg() again */
     }
     process_status = processMsg(msg);
     ReplyMsg(msg);
     if(process_status == QUIT) {
        break;
     }
}
```

It is possible to extend this loop to service events from more than one message port:

```
     signal = Wait(signal1 |  signal2  | ... signaln);
     if (signal & signal1) { /* do stuff for signal1 */ }
     if (signal & signal2) { /* do stuff for signal2 */ }
     ...
```

"signal" is what is stored in the tc_SigRecvd field of the Task block. It is a mask of all the signals received since Wait was called. "signal1 | signal2 .. etc." is the mask of signals which EXEC will wake us up for (tc_SigWait).

Before exiting your application, be sure you have replied to all messages waiting on your port:

```
     while (msg = GetMsg(Port)) {
        ReplyMsg(msg);
     }
```

Launching A Task Revisited

```
#include <exec/exec.h>
#include <proto/exec.h>

struct Task   *theTask = NULL;
char          *myTaskName = "myTask";
ULONG         myvar;

/* The SubTask waits on a signal from the main program */
void myTask()
{
       struct MsgPort *MainTaskPort = NULL;
       struct Message MyMsg;

#ifdef AZTEC_C
       geta4();
#endif

       if (MainTaskPort = FindPort("Rendezvous") == NULL)
          exit(RETURN_FAIL);

       myvar = 100;

       PutMsg(MainTaskPort, &MyMsg); /* Could have just used Signal */

       (void) Wait(0L); /* Wait to be removed */
}

void main(int argc, char **argv)
{
       struct MsgPort *TaskPort = NULL;
       struct Message *Msg;
       myvar = 0;

       if (TaskPort = CreatePort("Rendezvous", 0L) == NULL)
          exit(RETURN_FAIL);

       theTask = CreateTask(myTaskName, 0L, myTask, 4000L);

       /* Must synchronize with SubTask first!*/
       Msg = WaitPort(TaskPort);

       printf("myvar is: %ld\n", myvar);

       if (TaskPort) DeletePort(TaskPort);

       /* We know it's safe to remove theTask */
       if (theTask) DeleteTask(theTask);
}
```

EXEC I/O

```
struct IORequest {
     struct Message io_Message;   /* used by device to return IOReq */
     struct Device *io_Device;    /* Device Node pointer */
     struct Unit *io_Unit;        /* Unit (driver private */
     UWORD io_Command;            /* Device command */
     UBYTE io_Flags;
     BYTE io_Error;               /* Error or Warning # */
};
```

OpenDevice(devName, Unit, ioRequest, Flags)
- Open a named device (device names follow the convention "name.device" and live in the DOS DEVS: directory or in ROM) and initialize the IORequest block.

CloseDevice() - close a given device.

All the following functions take an IORequest block as an argument:

SendIO() - initiate an IO command and return immediately (asynchronous)

DoIO() - perform an IO command and wait for completion (synchronous)

CheckIO() - check to see if an IO Request has completed

WaitIO() - wait for the completion of an IO Request

AbortIO() - attempt to abort an IO Request in progress

BeginIO() - perform an IO command either synchronously or asynchronously

I/O completion tells you that an IORequest block may be used again. So the Reply is implicit upon I/O completion.

IORequest Blocks

```
struct IOStdReq {
     struct Message io_Message;
     struct Device  *io_Device;
     struct Unit    *io_Unit;
     UWORD          io_Command;
     UBYTE          io_Flags;
     BYTE           io_Error;
     ULONG          io_Actual;  /* Actual length that occured (bytes) */
     ULONG          io_Length;  /* The length of this requested I/O */
     APTR           io_Data;    /* Pointer to data of size io_Length */
     ULONG          io_Offset;  /* Offset for block structured devices */
}
```

Various device drivers require variations on the basic IORequest block. The following compiler library routines are available to allocate and free some of the more commonly used IORequest structures:

```
struct IOStdReq *CreateStdIO(port)
     struct MsgPort *port;

DeleteStdIO(iop)
     struct IOStdRequest *iop;

struct IOStdReq *CreateExtIO(port, length)
     struct MsgPort *port;
     ULONG length;

DeleteExtIO(iop)
     struct IOStdReq *iop;
```

Always do I/O in the largest chunks possible. This will reduce the amount of overhead placed on the system by the Operating System.

Device Example - Using The timer.device

```
#include <exec/types.h>
#include <exec/memory.h>
#include <proto/exec.h>
#include <devices/timer.h>

#define WAIT_TIME     100000L  /* Timer delay in microseconds */

extern ULONG               TimerBits;

struct MsgPort             *Timer_Port = NULL;
struct timerequest         *Time_Req = NULL;
BOOL                       topen = FALSE;

InitTimer()
{
      if (!(Timer_Port = CreatePort("Timer Port", 0L))) return(0);
      if (!(Time_Req = (struct timerequest *)
         CreateExtIO(Timer_Port,(long) sizeof(struct timerequest)))) {
         KillTimer();
         return(0);
      }
      if (OpenDevice(TIMERNAME, UNIT_VBLANK, Time_Req, 0L)) {
         KillTimer();
         return(0);
      }
      topen = TRUE;
      TimerBits = (1L << Timer_Port->mp_SigBit);
      return(1);
}

QueueTime()
{
      Time_Req->tr_node.io_Command = TR_ADDREQUEST;
      Time_Req->tr_time.tv_secs = 0L;
      Time_Req->tr_time.tv_micro = WAIT_TIME;
      SendIO(Time_Req);
}

KillTimer()
{
      if (topen) {
         AbortIO(Time_Req);
         WaitIO(Time_Req);
         CloseDevice(Time_Req) ;
      }
      if (Time_Req) DeleteExtIO(Time_Req);
      if (Timer_Port) DeletePort(Timer_Port) ;
}
```

Example - timer.device (cont'd)

```
#include <exec/exec.h>
#include <proto/exec.h>
#include <time.h>

extern struct MsgPort      *Timer_Port;
extern struct timerequest  *Time_Req;

ULONG        TimerBits;
LONG         tloc, chip_free, fast_free;
char         date[24] = NULL;

/* This procedure relies on the standard C library routine ctime() */
GetTime()
{
      tloc = time(0L);
      strcpy(date, ctime(&tloc));
      chip_free = AvailMem(MEMF_CHIP);
      fast_free = AvailMem(MEMF_FAST);
}

main()
{
      BOOL Done = FALSE;
      ULONG signals, times = 0;

      if (!InitTimer()) exit();

      GetTime();          /* Get the initial time */
      QueueTime();        /* Queue up a timer request */

      while (!Done) {
         signals = Wait(TimerBits);

         if ((signals & TimerBits) != 0L) {
            WaitIO(Time_Req); /* Wait for IO to complete, and remove Msg */
            times++;
            if (times == 10) {
                  Done = TRUE;
                  continue;
            }
            GetTime();     /* Get the current time */
            QueueTime();   /* Queue up a new timer request */
         }
      }
      KillTimer();
}
```

Guru Meditations (1.3)

We all get them sometime, but what are they?

The Alert() mechanism is in place to warn the user of potential software or hardware problems

An Alert() can be "Recoverable" or "Dead End"

A Guru Meditation Alert is almost always a Dead End Alert

The various Amiga-specific alert codes are described in the include file, exec/alerts.h. Low numbered Alerts (particularly 2-11,32-47) are standard 68000 exceptions.

A typical Guru Alert looks like this:

80038007.nnnnnnnn

The first number represents either the 68000 exception vector or the Kernel subsystem which failed. The second number represents the address at which the error occured. The first number may actually be a combination of numbers OR'ed together, so this alert means:

80000000 = This is a Dead End Alert
00030000 = OpenLibrary error
00008007 = on dos.library

(This error would certainly be disastrous, and will probably never happen on your system).

Your own application may put up an Alert using the EXEC function Alert(), or the Intuition function DisplayAlert().

Alert() takes two parameters:

Alert(alertNum, parameters)

alertNum == exception vector or subsystem, parameters == taskid or PC

System Errors (2.0)

Same system call as 1.3 (Alert), one parameter:

Alert(alertNum)

The format of alertNum remains the same

Now called "System Error"

Yellow indicates recoverable alert

Red indicates non-recoverable alert

Alerts now appear as:

Error: 0000 0000 Task: 00000000

Where the Error is split into the subsystem #/general error and the specific failure code and Task represents the task ID or PC at the time of failure

For Alerts caused by 680x0 exceptions, all registers are copied to a special place in low memory

Certain Alerts which were non-recoverable in 1.3 are recoverable in 2.0 (e.g. FreeTwice)

Tips And Caveats

1. Never busy wait - always use Wait()/WaitIO()/WaitPort().

2. Never allocate a resource you will not use, and return all resources as quickly as possible.

3. Cast return values and parameters to their proper data types.

4. Always check your return values!

5. When in doubt, consult the AutoDocs.

6. Follow every allocation with a deallocation, every open with a close, every forbid with a permit, etc. etc.

7. Never hardcode a ROM call or the location of a hardware resource. Always use the Libraries to make your calls and Resources to find hardware.

8. If something fails, notify the user and exit gracefully.

9. Close resources in the reverse order of allocation.

10. Use whatever tools speed your work or otherwise enhance your environment.

10. Be Consistent!

Topics For Further Study

Devices - console device, serial device, audio device, printer device.

Libraries - diskfont.library, math libraries

Memory Management - CopyMem(), CopyMemQuick().

MultiTasking - Semaphores

Utilities - Hooks, Tags

References Used To Prepare This Course

1. "Amiga ROM Kernel Reference Manual: Libraries and Devices",
   Commodore-Amiga, Inc., Addison Wesley
2. "Amiga ROM Kernel Reference Manual: Includes and Autodocs",
   Commodore-Amiga,Inc., Addison Wesley
3. "Amiga Hardware Reference Manual",
   Commodore-Amiga, Inc., Addison Wesley
4. Autodocs, Commodore-Amiga Inc.
5. Amiga Developer Conference Notes, 1988-1990, Commodore-Amiga Inc.
6. Manx Aztec C Library Source, Manx Software Systems Inc.
7. Motorola M68000 Microprocessor Reference Manual, Motorola Microsystems Inc.
8. The Fred Fish Public Domain Disk Library, Fred Fish

The Autodocs and Developer Conference Notes can be purchased directly from Commodore-Amiga:

Commodore-Amiga Technical Support
1200 Wilson Dr.
West Chester, PA 19380
1-215-431-9180

Manx Aztec C
Manx Software
1 Industrial Way
Eatontown, NJ 07724
1-908-545-2121

SAS/C
SAS Institute, Inc.
SAS Campus Drive.
Cary, NC 27513
1-919-677-8009

Fish Disks
Fred Fish
1346 W. 10th Pl
Tempe, AZ 85281

The ROM Kernel manuals can be purchased at most B. Dalton's bookstores or at your local Amiga dealer (support your local dealers!!!)

Also recommended:

The AmigaDOS Manual
Bantam Books, Commodore-Amiga

The Kickstart Guide To The Amiga
Ariadne Software

Programmers Guide To The Amiga
Rob Peck
Sybex Publishing

Fundamentals of Interactive Computer Graphics
J. D. Foley, A. Van Dam

Graphics in Overlapping Bitmap Layers
Rob Pike
ACM Transactions on Graphics 1983

Your Local User Group - Support It!

Course Evaluation

Date __/__/__

Student Name: (Optional) ________________________

Course Content

How well did the content meet the objectives of the course? 1 2 3 4 5 6 7 8 9 10
Not Very Very Well

Please list topics that you felt were:
a) presented in too much detail b) presented in too little detail
c) presented but did not apply d) not presented but should have been

Topic:	Comment
________________________	a b c d
________________________	a b c d
________________________	a b c d
________________________	a b c d

Were the course materials easy to read and follow? __Yes __ No __ Undecided

Please rate your overall satisfaction with the content 1 2 3 4 5 6 7 8 9 10
Least Most

Instructor

Did the instructor use examples that reinforced the course material? __ Yes __ No __ Undecided
Did the instructor summarize the material to emphasize the important points? __ Yes __ No __ Undecided
Did the instructor demonstrate a good understanding of the course content? __ Yes __ No __ Undecided
Please provide any other comments you may have about the instructor: ________________

____

____

Please rate your overall satisfaction with the instructor 1 2 3 4 5 6 7 8 9 10
Least Most

Would you recommend this course to an associate? __ Yes __ No __ Undecided

Are there other topics you would like to see addressed in this format?

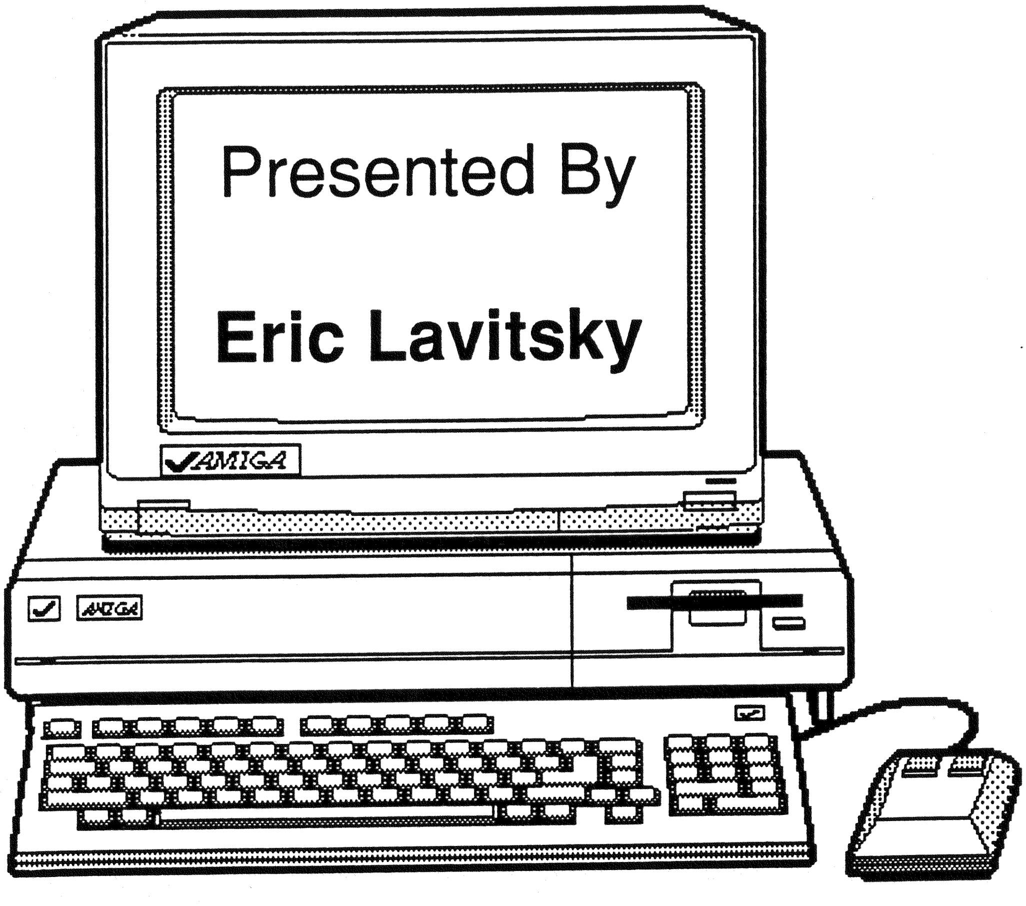
____

____

____

Thank you for taking the time to complete this survey. Your opinions are valuable and will help me to improve the quality of the material I present.. Please place any additional comments you may have on the reverse side of this sheet.

Introduction To Programming The Amiga®
Part II



Presented By

Eric Lavitsky

Copyright © 1988-1991 by Eric Lavitsky
All Rights Reserved Worldwide

**This course is the exclusive property of the author,
Eric Lavitsky, 174 Hyde Park Rd., Somerset, NJ 08873 (908) 560-0106
and may not be reproduced in any way without express written permission.**

This course is dedicated to all the wonderful friends and accquaintances I have made over the past few years working with the Amiga and to everyone who made the Amiga a reality.

Amiga is a registered trademark of Commodore-Amiga Inc.

This document was created entirely on an Amiga A2000/A2500 using Professional Page from The Gold Disk Inc. and printed on a PostScript Laser Printer.

What Are The Goals Of This Course?

- To provide an understanding of fundamental concepts of Amiga architecture and programming.
- To provide the basic tools to begin writing Amiga applications.
- To provide insight into elements of good (and bad) Amiga programming style.

Prerequisites:

This course assumes a knowledge of 'C' or a similiar structured programming language and an understanding of basic concepts in computer architecture. A familiarity with assembly language and other multi-tasking/multi-processor environments is not required, but will be helpful.

Disclaimer: Many of the examples in this course are presented as fragments or pseudo-code. They were derived from various C Compilers/Assemblers and tested wherever possible. The author has made every attempt to verify their accuracy, but assumes no liability for their content.

Course Outline

Graphics

Overview

- Terminology
- Display Overview
- Graphics Hierarchy (Views, BitMaps, RastPorts)
- EHB and HAM Mode
- ECS Modes
- Overscan
- BitMaps and Rasters

RastPorts

- The RastPort Structure
- Pens and Draw Modes
- Drawing in Large Areas

Layers

- Clipping Regions/Damage Lists

Intuition

- Screens
- Windows
- Calling graphics.library Routines - Lines and Text
- Intuition Rendering
- The IDCMP
- Events in Intuition - Wait() and The IDCMP
- Window Refresh Events
- Menus and Menu Events
- Gadgets
- Requesters
- Keyboard Processing

AmigaDOS

- Overview
- File I/O
- Workbench/CLI Environment

Graphics Terminology

BitBlt

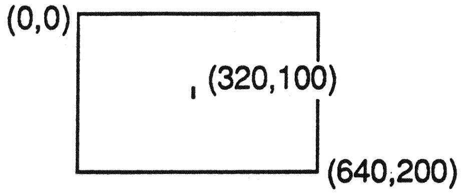
Short for Bit-Block Transfer. This routine will quickly and efficiently copy one source rectangular region to a destination rectangular region. The Amiga has special hardware to perform this operation, known as the Blitter (or Bimmer).

BitMap

A region of memory which defines the bounds and makeup of the display.

Cartesian Coordinate System

The standard mathematical coordinate system for a 2-D surface. Every pair (x,y) maps directly to a point (x,y) in the target space:



Clipping

The process through which drawing primitives whose coordinates lie outside the viewable portion of a "window" are cut off and not mapped to the display.

Double Buffering

Used to create smooth animation. Two seperate displays are maintained. While one frame is being displayed, rendering occurs in the second, offscreen display.

Half-Bright

Or "Extra Half-Bright". Denise can generate a second 32 colors from the original pallete of 32 using this special technique. The new colors will be a modification of the original 32.

HAM

Short for Hold-and-Modify. This special graphics mode allows 4096 colors to be displayed in the same View at once.

Graphics Terminology (contd.)

Interlace

The technique whereby two succesive video fields are combined into one frame for display.

Overscan

The region of the display normally hidden from view which allows the display to fill to the edges of the display hardware.

Pixel

Or Picture Element an individual (x,y) pair within a BitMap.

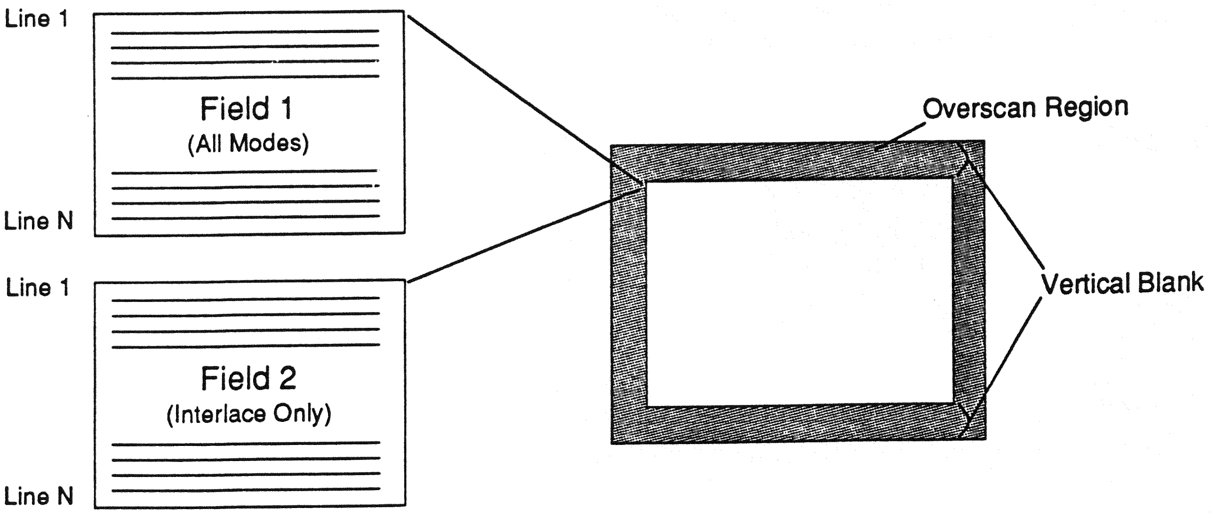
Raster

Also refered to as a Scan Line. One row of the display. Also used to describe the entire display made up of a matrix of pixels.

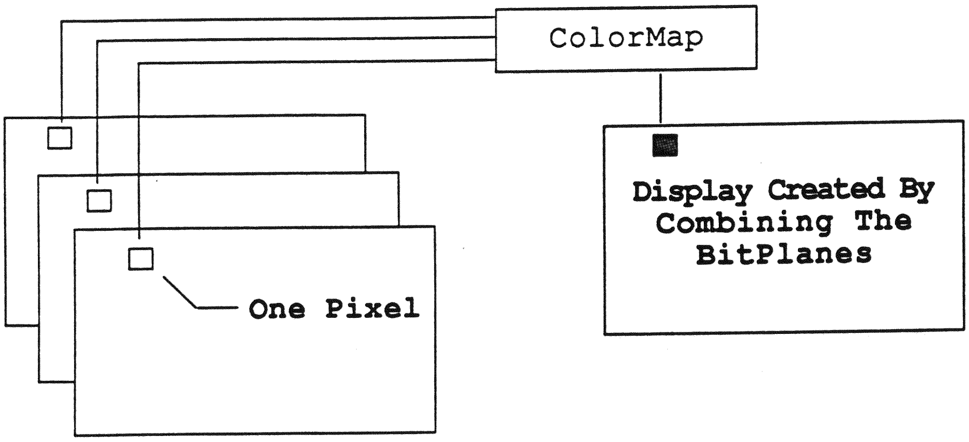
Sprite

Hardware objects which can be moved independently of the background View, 16 pixels wide and as high as you like. The mouse pointer is a sprite. The hardware can also detect when sprites "collide" with each other or other parts of the display.

Hardware Display Overview


Line 1
Field 1
(All Modes)
Line N
Line 1
Field 2
(Interlace Only)
Line N
Overscan Region
Vertical Blank

Display Overview



ViewPort

A ViewPort is the lowest level most programmers will deal with unless they actually play with Copper Lists. A View has a specific resolution, size and ColorMap associated with it.

BitMap

This region of memory has an associated depth which corresponds to the number of BitPlanes and hence the number of colors able to be displayed.

RastPort

This is the data structure most graphics routines use for rendering. The underlying mechanism writes the data (through a Layer) to a BitMap associated with a RastPort.

Layer

When placed on top of a RastPort's BitMap, a layer will clip all rendered graphics to the bounds of the Layer rectangle.

Screen

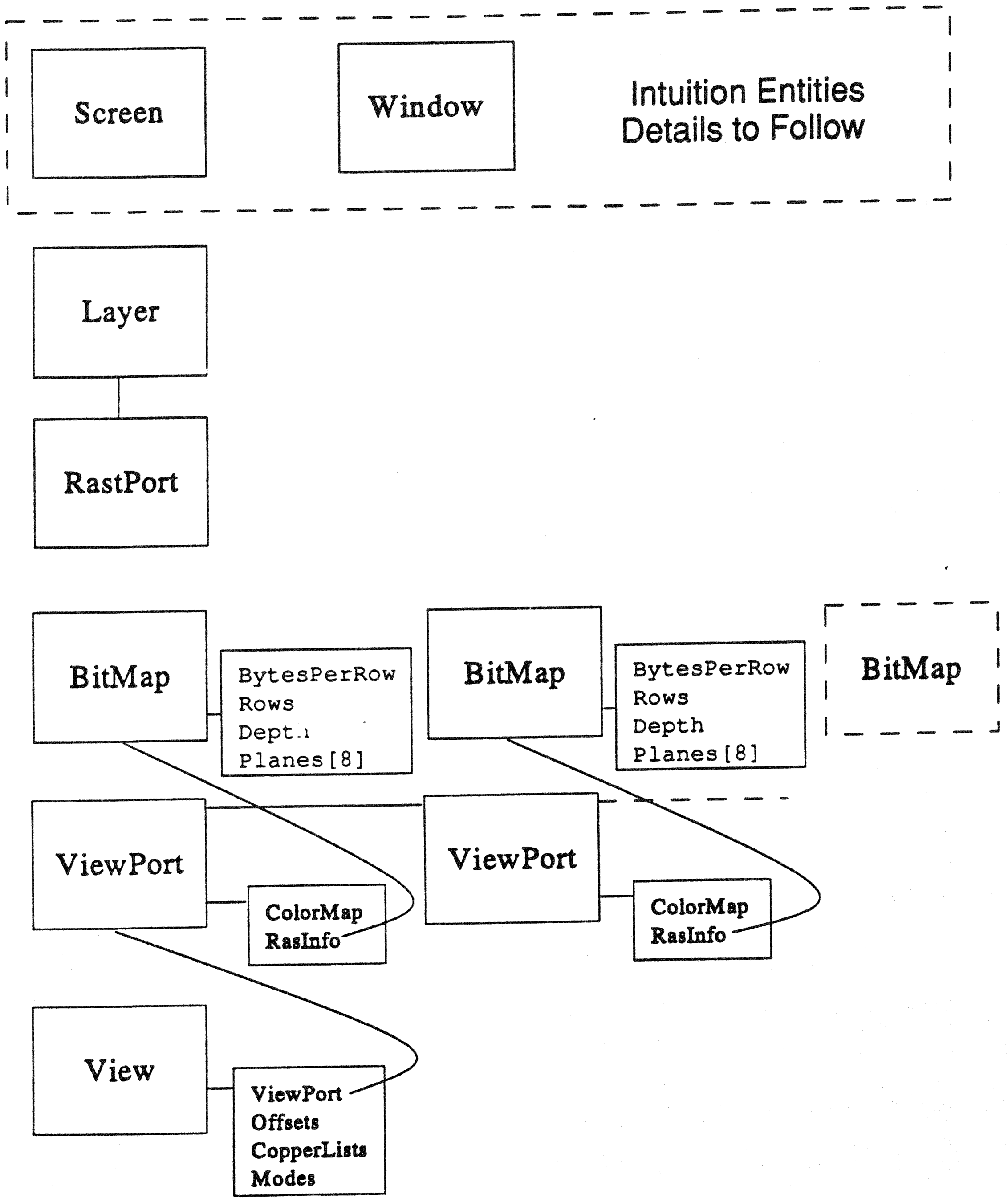
An Intuition entity. A Screen has an underlying RastPort and View (hence a specific resolution). Screens have "Drag Bars" and "Front/Back Gadgets".

Window

Is a RastPort with a Layer plus a drag bar, front/back gadgets, border, perhaps menus attached to it. It lives on a Screen. Windows can accept user input via the keyboard and mouse.

Note that all these data structures are essentially rectangular regions.

The Graphics Hierarchy



Screen

Window

Intuition Entities
Details to Follow

Layer

RastPort

BitMap

BytesPerRow
Rows
Depth
Planes[8]

BitMap

BytesPerRow
Rows
Depth
Planes[8]

BitMap

ViewPort

ColorMap
RasInfo

ViewPort

ColorMap
RasInfo

View

ViewPort
Offsets
CopperLists
Modes

EHB and HAM Mode

Available in low-resolution only

EHB: Extra Half Brite. Provides second set of 32 colors on a low-resolution, 5-bitplane display. The second set of 32 colors is automatically derived as half the intensity of the first 32 colors in the palette. `ViewPort->Modes = EXTRA_HALF_BRITE;`

HAM: Hold and Modify. Allows dynamic color table modification to achieve more simultaneous colors ona low-resolution display. The output color value is held, one of the color components is modified based on the value contained in bit-planes 5 and 6, and the resultant pixel value is then output to the display: `ViewPort->Modes = HAM;`

> **If bit-planes 5 and 6 contain 0, then bit-planes 1-4 contain the actual color register (0-15) to use for the pixel.**
>
> **If plane 6 is "0" and plane 5 is "1", the color of the pixel immediately to the left is duplicated and then modified. The bits from planes 4 to 1 are used to replace the four "blue" bits.**
>
> **If plane 6 is "1" and plane 5 is "0", the color of the pixel immediately to the left is duplicated and then modified. The bits from planes 4 to 1 are used to replace the four "red" bits.**
>
> **If plane 6 is "1" and plane 5 is "1", the color of the pixel immediately to the left is duplicated and then modified. The bits from planes 4 to 1 are used to replace the four "green" bits.**

ECS Modes

Denise Features:

Productivity modes: standard resolutions, non-interlaced in 4 out of 64 colors. Useful for DTP/Business applications.

Super-HiRes Mode: 35ns pixel mode (as opposed to 70ns standard pixel). Offers 1200x400 in 4 out of 64 colors. Useful for video-titling.

Genlock Features

Key to any color or bitplane
Genlock to border only

Agnus Features:

Big blits (up to 32767 x 32767)
Address more Chip RAM (up to 2Meg)

Performance Issues:

Productivity modes severely impact performance
Super-HiRes mode impacts performance

Overscan

Purpose: overscan allows video to be displayed "edge to edge" on standard broadcast displays. The Amiga hardware supports the following overscan modes:

Vertical Resolution:

	NTSC	PAL
Non-Interlaced	241	283
Interlaced	482	566

Horizontal Resolution is a maximum of 362 pixel in low-resolution or 724 pixels in high-resolution (both NTSC and PAL).

1.3 Issues

- Not supported in OS
- Requires poking GfxBase
- Refer to Jim Mackraz's DevCon '90 notes

2.0 Issues

- Supported in OS
  - functions:
    - OpenScreen()
    - QueryOverscan()
  - types:
    - standard overscan
    - text overscan
    - max overscan
    - video overscan

Performance Issues

- Increased chip memory usage
- Increased saturation of chip busbandwidth
  - hi-res + overscan
  - deep pixels

BitMaps and Rasters

InitBitMap(Bmap, NPlanes, Width, Height);

Where Bmap is a pointer to a region of memory large enough to hold the bitmap ((Width*Height)/8 *NPlanes),
NPlanes is the number of Bit-Planes in this bitmap (the Depth), and Width and Height are the x and y dimensions (in pixels) of the bitmap.

Then for each Bit-Plane n:

Bmap->Planes[n] = AllocRaster(Width, Height);

Don't forget to free your memory!
For all Bit-Planes n:

FreeRaster(BMap->Planes[n], Width, Height);

RastPorts

```
struct RastPort
{
        struct Layer *Layer;           /* The layer for clipping */
        struct BitMap *BitMap;         /* The actual BitMap */
        USHORT *AreaPtrn;              /* ptr to AreaFill pattern */
        struct TmpRas *TmpRas;         /* temp storage for rendering */
        struct AreaInfo *AreaInfo;     /* for area fills */
        struct GelsInfo *GelsInfo;     /* for GELS system
        UBYTE Mask;                    /* Write Mask for this raster */
        BYTE FgPen;                    /* ForeGround pen */
        BYTE BgPen;                    /* BackGround pen */
        BYTE AOlPen;                   /* AreaFill Outline pen */
        BYTE DrawMode;                 /* Drawing mode for fill,lines,text */
        BYTE AreaPtSz;
        BYTE linpatcnt;
        BYTE dummy;
        USHORT Flags;
        USHORT LinePtrn;
        SHORT cp_x, cp_y;              /* current pen position */
        UBYTE minterms[8];
        SHORT PenWidth;
        SHORT PenHeight;
        struct TextFont *Font;         /* current font */
        UBYTE AlgoStyle;               /* the algorithmic font style */
        UBYTE TxFlags;                 /* Text flags */
        UWORD TxHeight;                /*  Text height */
        UWORD TxWidth;                 /* text width */
        UWORD TxBaseline;              /*  text baseline */
        WORD TxSpacing;                /* text spacing (per character)  */
        APTR *RP_User;
        ULONG longreserved[2];
#ifndef GFX_RASTPORT_1_2
        UWORD wordreserved[7];
        UBYTE reserved[8];
#endif
};
```

To initialize and draw into a RastPort you must first open the graphics.library.

Pens and Draw Modes

The graphics primitives which work within RastPorts can also have three different drawing "pens" associated with tem. They are:

FgPen - the foreground/primary pen. Generally used for rendering lines and areas.

BgPen - the background/secondary pen.

AOlPen - the area outline pen.

Every RastPort has an associated drawing mode. This DrawMode determines

JAM1 - one color is "jammed" (rendered) into the target area. The color used is the FgPen

JAM2 - two colors are "jammed" into the target area.Wherever there is a '1' bit, use the FgPen, wherever there is a '0' bit, use the BgPen.

COMPLEMENT - each bit value is reversed (complemented). This is an exclusive or (XOR) and useful for drawing and then erasing lines/points. (Note: all bits in a pixel are complemented - the FgPen is ignored. Use the RPort->Mask to set the bitplanes affected by COMPLEMENT).

INVERSVID - used primarily for text, when combined with JAM1 the text appears as a transparent letter surrounded by the FgPen color. If JAM2 is set, BgPen is used to draw the character itself.

InitRastPort(rp) - initialize a RastPort structure.
SetAPen(rp, n) - change the color for the FgPen to color register n (32 max).
SetBPen(rp., n) - change the color for the BgPen to color register n (32 max).
SetOPen(rp., n) - change the color for the AOlPen to color register n (32 max).
SetDrMd(rp, mode) - change the current drawing mode.
SetAfPt(rp, apt, power_of_2) - change the current area pattern.
SetDrPt(rp, n) - change the current line drawing pattern (n =0xffff = solid).
Move(rp, x, y) - moves the pen in RastPort (rp) to a new position (x,y).
Draw(rp, x, y) - draws a line from the current pen position to (x,y).
PolyDraw(rp, count, arrayptr) - draws a shape using a set of line endpoints.
ReadPixel(rp, x, y) - returns the value (color) of the pixel at (x,y).
WritePixel(rp, x, y) - changes the pixel at (x,y) using the FgPen.

Drawing In Larger Areas

AreaMove(rp, x, y) - begin a new polygon at (x,y).
AreaCircle(rp, cx, cy, radius)/AreaEllipse(rp, cx, cy, a, b) - add a circle or ellipse to the RastPort AreaInfo.
AreaDraw(rp, x, y) - add a new vertex to the list within the current polygon.
AreaEnd(rp) - draw all the shapes in the current polygon and fill them. This routine will do it's rendering using the curent line and area pattern. It will also automatically close the polygon if the first and last vertices do not meet.
BltClear(memb, bytecnt, flags) - clear the chip memory pointed to by memb of length bytecnt.
BltBitMap()/ClipBlit(rp, x, y, urp, xd, yd, sizex, sizey, minterm) - copy a rectangular area from one section of chip memory to another using the logic operation defined in minterm. ClipBlit() works in RastPorts and Layers.
BltPattern(rp, mask, xl, yl, maxx, maxy, bytecnt) - draw to a rectangular region through a stencil specified by mask.
Flood(rp, mode, x, y) - flood fill an area.
RectFill(rp, xmin, ymin, xmax, ymax) - fill a rectangular area.
SetRast(rp, pen) - set an entire RastPort to the color of pen.

Some operations require that a TmpRas be allocated and attached to your RastPort->TmpRas (most Area and Flood routines).

```
PLANEPTR myplane;
struct TmpRas mytmpras;

myplane = AllocRaster(XSIZE, YSIZE);
if (myplane == 0) exit(1);
rp->TmpRas = InitTmpRas(&mytmpras, myplane, RASSIZE(XSIZE, YSIZE));
```

The Area routines also require that you initialize the RastPort AreaInfo using InitArea():

```
        InitArea(rp, areabuffer, count);
```

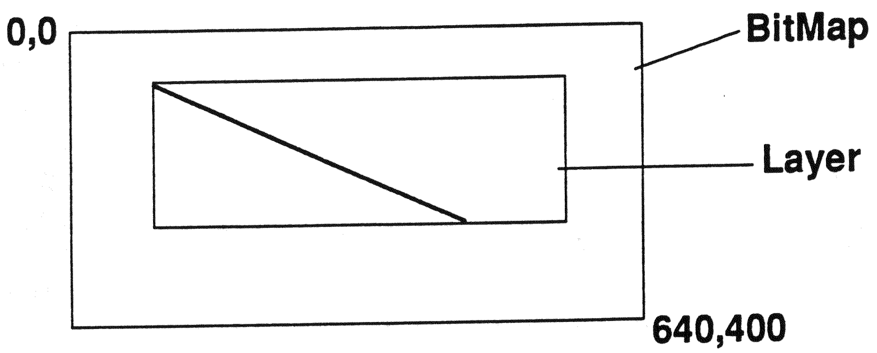
areabuffer must point to an array which allows 5 bytes per vertex in your area on a WORD boundary. Therefore, to initialize an area which can handle 100 vertices:

```
        UWORD areabuffer[250];
        InitArea(rp, &areabuffer[0], 100L);
```

Layers

Allow more information to be displayed on a small screen area.

Allow clipping to arbitrary rectangular regions using ClipRects.



So, in our diagram above, if we draw a line from 0,0 to 640,400 it get's clipped to the bounds of our Layer.

The underlying data structure of a **Layer** is a **ClipRect**, and underneath that is a **Rectangle**.

If a Layer is covered by another Layer, it is said to be obscured.

Obscured regions are maintained in a Damage List so that they may be properly re-created when they are brought back into view.

If a Layer is SmartRefresh, this re-creation is done for you at the expense of some memory overhead.

If a Layer is SimpleRefresh, your application will be notified of the need to refresh portions of your display.

A single BitMap may have an arbitrary number of Layers associated with it.

Intuition

The User Interface

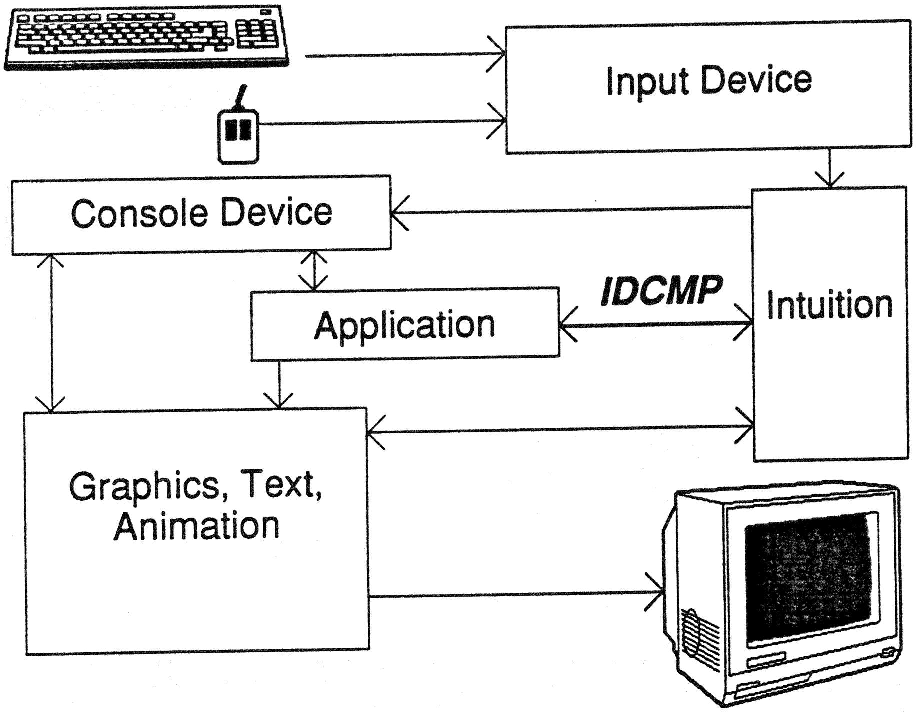
- Intuition manages input devices such as the mouse and keyboard through the "input.device"

- Intuition takes the events generated by these devices and takes action on them.

- Intuition can output to the display based on these events or under application program control.

- Intuition can communicate events to a specific task which has an Intuition Direct Communication Message Port associated with it (IDCMP)

This is sometimes referred to as the Intuition "Food Chain"


Input Device
Console Device
IDCMP
Application
Intuition
Graphics, Text, Animation

Screens

Screens are Views with a BitMap and RastPort attached as well as optional drag and front/back gadgets along the top.

A Screen has a specific resolution associated with it just like a View.
For example, the WorkBench Screen has a resolution of 640x200.

```
struct NewScreen
{
      SHORT LeftEdge, TopEdge, Width, Height, Depth;
      UBYTE DetailPen, BlockPen;
      USHORT ViewModes;          /* The modes for the ViewPort */
      USHORT Type;               /* The Screen type (see below) */
      struct TextAttr *Font;     /* The requested default font */
      UBYTE *DefaultTitle;       /* The default title */
      struct Gadget *Gadgets;    /* not supported */
      struct BitMap *CustomBitMap; /* own BitMap for CUSTOMSCREEN  */
};
```

The following Flags describe the Screen Type:

WBENCHSCREEN - This Screen is the WorkBench Screen (you cannot request this).
CUSTOMSCREEN - This Screen is independent of the WorkBench display.
SHOWTITLE - Determines whether or not the Screen title bar is displayed.
(this flag is set by a call to Intuition routine ShowTitle())
CUSTOMBITMAP - This allows you to supply your own BitMap for the Screen.
SCREENBEHIND - Open this Screen behind all other Screens instead of in front.
SCREENQUIET - Do not render the imagery for the Screen gadgets.
(additional 2.0 flags are available through OpenScreenTags())

The following ViewModes are legal under both 1.3 and 2.0:

- **HIRES** - selects high resolution (640 pixels across).
- **INTERLACE** - selects interlace mode (400 lines).
- **SPRITES** - set this if you want to use sprites on your display.
- **DUALPF** - set this flag if you want two playfields.
- **EXTRA_HALFBRITE** - set this flag if you want extra-halfbrite (64 colors).
- **HAM** - set this flag if you want hold-and-modify (4096 colors).

(additional 2.0/ECS modes are available through OpenScreenTags())

Note that HAM and EXTRA_HALFBRITE cannot be used in conjunction with HIRES.

Screens in Detail

```
struct Screen {
      struct Screen *NextScreen;      /* ptr to next screen */
      struct Window *FirstWindow;     /* list of windows */
      SHORT LeftEdge, TopEdge;        /* */
      SHORT Width, Height;            /*  */
      SHORT MouseY, MouseX;           /* pos relative to upper left */
      USHORT Flags;                   /* See screen type flags */
      UBYTE *Title;                   /*  The screen's initial title */
      UBYTE *DefaultTitle;            /* The default title for windows */
      BYTE BarHeight, BarVBorder, BarHBorder, MenuVBorder,  MenuHBorder;
      BYTE WBorTop, WBorLeft, WBorRight, WBorBottom;
      struct TextAttr *Font;          /* The screen's default font */
      struct ViewPort ViewPort;       /* The screen's display */
      struct RastPort RastPort;       /* The screen's rastport */
      struct BitMap BitMap;           /*  */
      struct Layer_Info LayerInfo;
      struct Gadget *FirstGadget;     /* List of user-supplied gadgets */
      UBYTE DetailPen, BlockPen;
      USHORT SaveColor0;
      struct Layer *BarLayer;         /* The menu/title/drag bar layer */
      UBYTE *ExtData;
      UBYTE *UserData;
};
```

OpenScreen() - pass this a pointer to an initialized NewScreen structure and it will return a pointer to a Screen.

CloseScreen() - pass this a pointer to your screen and it will be closed and all it's memory deallocated.

MoveScreen() - pass this a pointer to a screen and a DeltaX/DeltaY pair to move the screen coordinates (only vertical movements are recognized)

ScreenToBack() - pass this a pointer to a screen and that screen will be sent to the back of the display.

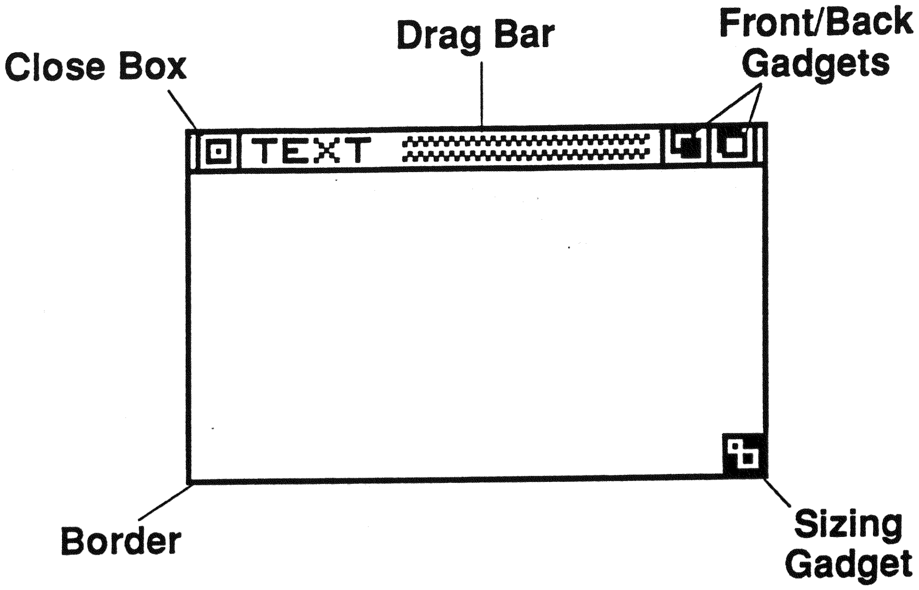
ScreenToFront() - pass this a pointer to a screen and that screen will be made the foremost in the display.

ShowTitle() - pass this a pointer to a screen and TRUE/FALSE to enable/disable the displaying of the screen title bar.

Opening A Screen

```
#include <exec/exec.h>              /* Include all the Exec Headers */
#include <intuition/intuition.h>    /* Include all the Intuition Structures */
#include <proto/exec.h>
#include <proto/intuition.h>

struct IntuitionBase    *IntuitionBase = NULL;
struct Screen           *s = NULL;
struct NewScreen        ns = {
        0, 0,                       /* Left Edge, Top Edge */
        640, 200, 2,                /* Width, Height, Depth */
        0, 1,
        HIRES,                      /* ViewModes */
        CUSTOMSCREEN,               /* Type */
        NULL,                       /* Default Font */
        (UBYTE  *)"My Screen",      /* Screen Title */
        NULL,                       /* Gadgets */
        NULL                        /* Custom BitMap */
};

main()
{
        int i;

        if (!(IntuitionBase = (struct IntuitionBase *)OpenLibrary("intuition.library", 33L))) {
                        cleanexit();
        }

        if (!(s = (struct Screen *)OpenScreen(&ns))) {
                        cleanexit();
        }

        /* Never do this in your own code! */
        for (i -1; i <  10000; i++)  {}

        cleanexit();
}

cleanexit()
{
        if (s) CloseScreen(s);
        if (IntuitionBase) CloseLibrary(IntuitionBase);
        exit(0);
}
```

Windows In Detail



The Close Gadget and Sizing Gadget can return Intuition events to your application.

A window is essentially a layer, RastPort and some gadgets and imagery.

As such, there are window operations which correspond directly to layer operations:

WindowToFront() - UpFrontLayer()
WindowToBack() - BehindLayer()
MoveWindow() - MoveLayer()
SizeWindow() - SizeLayer()

Windows can be:

SMART_REFRESH
NOCAREREFRESH
BACKDROP
GIMMEZEROZERO
SIMPLE_REFRESH
SUPERBITMAP
BORDERLESS

These correspond to Layer types except BORDERLESS, which prevents the Window border from being rendered and GIMMEZEROZERO which places the coordinates (0,0) at the position just below the title bar.

Opening A Window

```
#include <exec/exec.h>              /* Include all the Exec Headers */
#include <intuition/intuition.h>    /* Include all the Intuition Structures */
#include <proto/exec.h>
#include <proto/intuition.h>

struct IntuitionBase     *IntuitionBase = NULL;
struct Window            *w = NULL;
struct NewWindow         nw = {
      0, 0,
      320, 200,
      0, 1,
      NULL,                                  /* IDCMP Flags */
      ACTIVATE | SMART_REFRESH | NOCAREREFRESH,/* Window Flags */
      NULL, NULL,                            /* Gadget, Image */
      (UBYTE *)"My Window",                  /* Window Title */
      NULL,                                  /* Screen */
      NULL,                                  /* BitMap */
      0, 0,                                  /* MinWidth, MinHeight */
      0, 0,                                  /* MaxWidth, MaxHeight */
      WBENCHSCREEN                           /* Window Type */
};

main()
{
   int i;

   if (!(IntuitionBase = (struct IntuitionBase *)OpenLibrary("intuition.library", 33L))) {
       cleanexit();
   }

   if (!(w = (struct Window *)OpenWindow(&nw))) {
       cleanexit();
   }

   /* Never do this in your own code! */
   for (i -1; i <  10000; i++)  {}

   cleanexit();
}

cleanexit()
{
   if (w) CloseWindow(w);
   if (IntuitionBase) CloseLibrary(IntuitionBase);
   exit(0);
}
```

Refreshing

SMART_REFRESH

All obscured regions are maintained in offscreen regions automatically. When you combine this flag with NOCAREREFRESH, you never have to worry about updating your window after it has been obscure by other layers/windows etc.

NOCAREREFRESH

When this flag is set, refresh events will under no circumstances be sent for this window. You must set this flag if you don't intend to call BeginRefresh()/EndRefresh() when REFRESHWINDOW events occur.

SIMPLE_REFRESH

When you receive a message through the IDCMP of class REFRESHWINDOW, you must call BeginRefresh()/EndRefresh() to redraw the necessary parts of your window and end the refresh-needed state.

```
BeginRefresh(w)
/* Redraw the display */
EndRefresh(w, Complete)
```

If Complete is FALSE, this tells Intuition that there is more updating to be done (perhaps by another routine or task within your own application). This method is slower than SMART_REFRESH since you must redraw the damaged sections yourself, but it is more RAM efficient than.

SUPERBITMAP

Here you get your own BitMap (which you must allocate) to use as display memory. This BitMap can be as large or larger than the window and you never have to worry about redisplaying it. If your BitMap is larger than the displayable window, you can use ScrollLayer() to reveal different portions of it.

Drawing A Line

```
#include <exec/exec.h>                  /* Include all the Exec Headers */
#include <graphics/gfxmacros.h>         /* Include */
#include <intuition/intuition.h>        /* Include all the Intuition Structures */
#include <proto/exec.h>
#include <proto/graphics.h>
#include <proto/intuition.h>

struct GfxBase           *GfxBase = NULL;
struct IntuitionBase     *IntuitionBase = NULL;
struct Window            *w = NULL;
struct NewWindow         nw = {
      0, 0, 320, 200, 0, 1,
      NULL, ACTIVATE | SMART_REFRESH,
      NULL, NULL,
      (UBYTE  *)"My Window",
      NULL, NULL, 0, 0, 0, 0, WBENCHSCREEN
};
struct RastPort          *RPort;

main()
{
   int i;

   if (!(GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",33L))) {
       cleanexit();
   }

   if (!(IntuitionBase = (struct IntuitionBase *)OpenLibrary("intuition.library", 33Lçç))) {
       cleanexit();
   }

   if (!(w = (struct Window *)OpenWindow(&nw))) {
       cleanexit();
   }

   RPort = w->RPort;                     /* Copy for convenience */
   SetAPen(RPort, 1L);                   /* Choose the pen to draw with */
   Move(RPort, 0L, 8L);                  /* Start at upper left */
   Draw(RPort, 320L, 200L);              /* Draw a line to lower right */

   /* Never do this in your own code! */
   for (i = 1; i <  100000; i++) { }

   cleanexit();
}

cleanexit()
{
   if (w) CloseWindow(w);
   if (IntuitionBase) CloseLibrary(IntuitionBase);
   if (GfxBase) CloseLibrary(GfxBase);
   exit(0);
}
```

Printing Some Text

```
#include <exec/exec.h>                    /* Include all the Exec Headers */
#include <graphics/gfxmacros.h>           /* Include */
#include <intuition/intuition.h>          /* Include all the Intuition Structures */
#include <proto/exec.h>
#include <proto/graphics.h>
#include <proto/intuition.h>

struct GfxBase              *GfxBase = NULL;
struct IntuitionBase        *IntuitionBase = NULL;
struct Window               *w = NULL;
struct NewWindow            nw = {
      0, 0, 320, 200, 0, 1,
      NULL, ACTIVATE |  SMART_REFRESH,
      NULL, NULL,
      (UBYTE  *)"My Window",
      NULL, NULL, 0, 0, 0, 0, WBENCHSCREEN
};
struct RastPort             *RPort;

main()
{
   int i;

   if (!(GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",33L))) {
       cleanexit();
   }

   if (!(IntuitionBase = (struct IntuitionBase *)OpenLibrary("intuition.library", 33L))) {
       cleanexit();
   }

   if (!(w = (struct Window *)OpenWindow(&nw))) {
       cleanexit();
   }

   RPort = w->RPort;                      /* Copy for convenience */
   SetAPen(RPort, 1L);                    /* Choose the pen to draw with */
   Move(RPort, 0L, 8L);                   /* Start at near upper left */
   Text(RPort, "Testing", 7L);            /* Put text in the default DrawMode */

   /* Never do this in your own code! */
   for (i = 1; i <  100000; i++) { }

   cleanexit();
}

cleanexit()
{
   if (w) CloseWindow(w);
   if (IntuitionBase) CloseLibrary(IntuitionBase);
   if (GfxBase) CloseLibrary(GfxBase);
   exit(0);
}
```

Intuition Rendering

TextAttr - used to describe the attributes of a *desired* font (graphics.library).

```
struct TextAttr {
       STRPTR ta_Name;              /* Name of the desired font */
       UWORD ta_YSize;              /* Height of the desired font */
       UBYTE ta_Style;              /* Intrinsic font style */
       UBYTE ta_Flags;              /* Font Preferences and Flags */
};
```

IntuiText - a data structure used to help Intuition render text. Used extensively in Menus, Gadgets etc. or can be used with the PrintIText() routine. IntuiTextLength() will return the width in pixels of an IntuiText structure.

```
struct IntuiText
{
       UBYTE FrontPen, BackPen;            /* pen numbers for rendering */
       UBYTE DrawMode;                     /* Graphics draw mode */
       SHORT LeftEdge;                     /* Relative Left Edge */
       SHORT TopEdge;                      /* Relative Top Edge */
       struct TextAttr *ITextFont;         /* Desired font */
       UBYTE *IText;                       /* ptr to NULL terminated text */
       struct IntuiText *NextText;         /* ptr to next IntuiText */
};
```

Border - used for drawing a series of lines intended for use as a border.

```
struct Border
{
       SHORT LeftEdge, TopEdge;            /* relative left and top edge */
       UBYTE FrontPen, BackPen;            /* pen numbers for rendering */
       UBYTE DrawMode;                     /* draw mode for rendering */
       BYTE Count;                         /* Number of x,y pairs */
       SHORT *XY;
       struct Border *NextBorder;          /* ptr to next Border */
};
```

Image - used for simple rendering of image data to a RastPort.

```
struct Image
{
       SHORT LeftEdge, TopEdge;            /* relative left and top edge */
       SHORT Width;                        /* pixel size (though data is word aligned */
       SHORT Height, Depth;                /* pixel sizes */
       USHORT *ImageData;                  /* ptr to the word-aligned bits */
       UBYTE PlanePick, PlaneOnOff;
       struct Image *NextImage;
};
```

Intuition Rendering (cont'd.)

DrawBorder() is Similiar to PolyDraw():

```
extern struct RastPort *rp;
SHORT mypairs[] = ( 0, 0, 10, 0, 10, 10, 0, 10, 0, 0 };
struct Border MyBorder = { 0, 0, 1, 2, JAM1, 5, mypairs, NULL};

Move(rp, 100L, 100L);
DrawBorder(rp, &MyBorder, 0L, 0L);
```

PrintIText() is similiar to Text()

```
struct IntuiText MyIText = {
      2, 1, JAM1, 0. 0. NULL, (UBYTE  *) "TEST", NULL
};

PrintIText(w->RPort, &MyIText, 10L, 10L);
```

DrawImage()

This example will render the image of the window sizing box into your RastPort:

```
USHORT idata[] = {
      0xFFFF, 0xC0FF, 0xCCFF, 0xC003,
      0xFCF3, 0xFCF3, 0xFCF3, 0xFC03, 0xFFFF
};

struct Image MyImage[] = {
      0, 0, 16, 9, 1, &idata[0], 0x1, 0x0, NULL
};

DrawImage(w->RPort, &MyImage, 10L, 10L);
```

The last two parameters in all these calls represent additional offsets into the RastPort used for rendering. All of these drawing routines will clip to the boundaries of the RastPort's layer.

Changing Colors

To change colors, you must be able to access a ViewPort. Every ViewPort has a ColorMap associated with it:

```
struct ColorMap {
      UBYTE Flags;
      UBYTE Type;          /* if = 0, ColorTable = UWORDS xRGB */
      UWORD Count;         /* # of entries in this ColorMap */
      APTR ColorTable;     /* A ptr to the color table itself */
}
```

LoadRGB4() - load RGB color values from ColorMap into a ViewPort.
GetRGB4() - find out the value of an entry in a ColorMap.
GetColorMap() - allocate and initialize a ColorMap of a given size.
FreeColorMap() - free a given ColorMap.

To get a ViewPort pointer from a window or screen, we must use the Intuition function ViewPortAddress(). Note that this function must be passed a pointer to a window.

```
/* In this example, our screen is 3 bit-planes deep yielding 8 colors */

      USHORT colors[8] = {    0x05a, 0xfff, 0x006, 0x007,
                              0x444, 0xbbb, 0x000, 0xf00 };

      struct Window     *w;
      struct ViewPort   *view;

      view = ViewPortAddress(w);
      LoadRGB4(view, (UWORD *) colors, 8L);
```

The IDCMP

(Intuition Direct Communication Message Port)

If you specify any IDCMPFlags whenever you open a window, you automatically get an **IDCMP** attached to it. You may have valid reasons for not wanting an IDCMP, in which case you would specify NULL.

You can use **ModifyIDCMP()** to change the flags in an **IDCMP** or to attach a particular **IDCMP** to a window. This can allow one common **IDCMP** to be shared among many windows.

IDCMP messages come through the window->UserPort->mp_SigBit

```
struct IntuiMessage
{
     struct Message ExecMessage;     /* This is an EXEC Message */
     ULONG Class;                    /* See IDCMP  Flags */
     USHORT Code;                    /* Special Values */
     USHORT Qualifier;               /* InputEvent qualifier */
     APTR IAddress;                  /* Intuition addresses */
     SHORT MouseX, MouseY;           /* Mouse coordinates */
     ULONG Seconds, Micros;          /* Current system clock */
     struct Window *IDCMPWindow;     /* This window */
     struct IntuiMessage *SpecialLink;
};
```

IIDCMP Flags and Events available under 1.3 and 2.0:

ACTIVEWINDOW
CLOSEWINDOW
INACTIVEWINDOW
NEWSIZE
REFRESHWINDOW
SIZEVERIFY
VANILLAKEY
RAWKEY
MOUSEBUTTONS
MOUSEMOVE
DELTAMOVE
INTUITICKS

GADGETDOWN
GADGETUP
REQSET
REQCLEAR
REQVERIFY
MENUPICK
MENUVERIFY
NEWPREFS
DISKINSERTED
DISKREMOVED

WBENCHMESSAGE

Wait() and The IDCMP

```
#include <exec/exec.h>              /* Include all the Exec Headers */
#include <intuition/intuition.h>    /* Include all the Intuition Structures */
#include <proto/exec.h>
#include <proto/intuition.h>

struct IntuitionBase    *IntuitionBase = NULL;
struct Window           *w = NULL;
struct NewWindow        nw = {
      0, 0,
      320, 200,
      0, 1,
      CLOSEWINDOW,                              /* IDCMP Flags */
      WINDOWCLOSE | ACTIVATE | SMART_REFRESH,   /* Window Flags */
      NULL, NULL,                               /* Gadget, Image */
      (UBYTE  *)"My Window",                    /* Window Title */
      NULL,                                     /* Screen */
      NULL,                                     /* BitMap */
      0, 0,                                     /* MinWidth, MinHeight */
      0, 0,                                     /* MaxWidth, MaxHeight */
      WBENCHSCREEN                              /* Window Type */
};
ULONG WindowBits;

main()
{
   int i;

   if (!(IntuitionBase = (struct IntuitionBase *)OpenLibrary("intuition.library", 33L))) {
       cleanexit();
   }

   if (!(w = (struct Window *)OpenWindow(&nw))) {
       cleanexit();
   }
   WindowBits = 1L << w->UserPort->mp_SigBit;

   Wait(WindowBits);                        /* Wait for any signal from the window */

   cleanexit();
}

cleanexit()
{
   if (w) CloseWindow(w);
   if (IntuitionBase) CloseLibrary(IntuitionBase);
   exit(0);
}
```

Cleaning Up The IDCMP

After using an **IDCMP**, you should be sure that you have replied to all outstanding messages before shutting down your application.

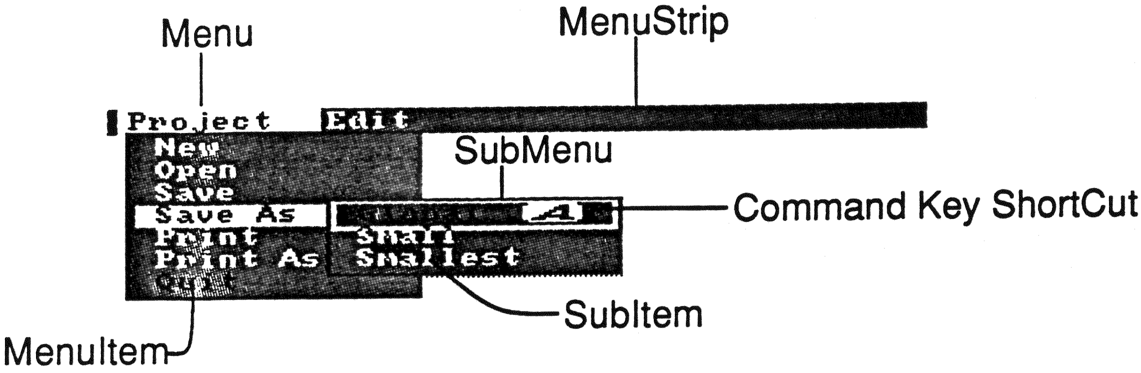
The proper way to be sure that all messages have been replied to, is to call the following routine before you close your window:

```
FlushIntuiMessages(w)
struct Window *w;
{
        register struct IntuiMessage *m;

        while (m = (struct IntuiMessage *) GetMsg(w->UserPort)) {
                ReplyMsg(m);
        }
}
```

Menus

Menus are not Windows or even Layers, they are clipping rectangles rendered with a special routine called SwapBitsClipRectRastPort().

When Menus are rendered, all other output to the display is halted.


Menu
MenuStrip
Project
Edit
New
Open
Save
Save As
Print
Print As
Quit
SubMenu
Command Key ShortCut
Small
Smallest
SubItem
MenuItem

Command Key ShortCuts will send the same event to your application as the normal menu selection would have.

Menus In Detail

The menu data structure:

```
struct Menu
{
        struct Menu *NextMenu;          /* same level */
        SHORT LeftEdge, TopEdge;        /* dimensions of the select box */
        SHORT Width, Height;            /* dimensions of the select box */
        USHORT Flags;                   /* */
        BYTE *MenuName;                 /* text for this Menu Header */
        struct MenuItem *FirstItem;     /* pointer to first in chain */
        SHORT JazzX, JazzY, BeatX, BeatY;
};

struct MenuItem
{
        struct MenuItem *NextItem;      /* pointer to next in chained list */
        SHORT LeftEdge, TopEdge;        /* dimensions of the select box */
        SHORT Width, Height;            /* dimensions of the select box */
        USHORT Flags;                   /* */
        LONG MutualExclude;             /* set bits mean this item excludes that */
        APTR ItemFill;                  /* points to Image, IntuiText or NULL  */
        APTR SelectFill;                /* points to Image, IntuiText or NULL  */
        BYTE Command;                   /* only if appliprog sets the COMMSEQ  flag */
        struct MenuItem *SubItem;       /* if non-zero, points to SubItem */
        USHORT NextSelect;              /*  */
};
```

SetMenuStrip(Window, Menu) - After filling in your Menu data structures, pass this routine a pointer to your window and a pointer to your first Menu. Your Menu is now attached to this Window.

ClearMenuStrip(Window) - You must call this routine before you close your Window.

OnMenu(Window, MenuNumber) - enable this Menu for selections by the user.

OffMenu(Window, MenuNumber) - disable this Menu. It will be "ghosted" out.

You must set the IDCMP Flag MenuPick if you wish to receive Menu events from Intuition.

Building A Menu

```
struct IntuiText Item1Text = {
        0, 1, JAM1, 1, 0, NULL, (UBYTE  *)  "Quit", NULL
};

struct MenuItem Item1 = {
        NULL,                        /* Next MenuItem in this Menu */
        0, 0,                        /* Add CHECKWIDTH to width if using CHECKIT
*/
        148 + COMMWIDTH, 10,         /* Add in the width of the command key */
        COMMSEQ |  ITEMTEXT  |  HIGHCOMP,
        NULL,                        /*  MutualExclude */
        &Item1Text,                  /* */
        NULL,
        'Q',                         /* Character to use for COMMSEQ */
        NULL,
        NULL
};

struct Menu MenuHead = {
        NULL,                        /* Next Menu in MenuStrip */
        0, 0,                        /* Dimensions of menu header box */
        78, 10,
        NULL,
        (UBYTE  *)  "Project",               /* Header text for this menu */
        &Item1,                      /* Pointer to first item in menu */
};

SetMenuStrip(w, &MenuHead);
```

You can build menus like this by hand (the fun way), or use any one of a number of commercial and public domain tools which offer a more WYSIWYG approach.

Processing Menu Events

```
extern struct Menu MenuHead;


/*
*       Our main Intuition state loop:
*       Call thusly: while (!GetIntuiMessages()) { Wait(... }
*/
int GetIntuiMessages()
{
        register struct IntuiMessage *m;
        register ULONG  Class = 0;
        register short  Code;

top:    if (m = GetMsg(lw->UserPort)) {
                Class = m->Class;
                Code  = m->Code;
                ReplyMsg(m);
        } else return(0);

        if (Class == MENUPICK) {
                if (!MenuSwitch(Code)) return(1);
        }
        goto top;
}

/*
*       Top level menu handler.
*/
MenuSwitch(code)
USHORT code;
{
        USHORT menunum;
        struct MenuItem *item;
        int error;

        error = TRUE;
        while(code != MENUNULL ) {
                item = (struct MenuItem *) ItemAddress(&MenuHead, code);
                menunum = MENUNUM( code );
                switch( menunum ) {
                        case 0:         /* Project */
                                error &= ProjectMenu(code);
                                break;
                        case 1:         /* Edit */
                                error &= EditMenu(code);
                                break;
                }
                code = item->NextSelect;
        }
        return(error);
}
```

Menu Events (contd.)

```
/*
*       Process selections from the Project menu.
*/
ProjectMenu(code)
USHORT code;
{
        USHORT itemnum, sub;

        itemnum = ITEMNUM (code);
        sub = SUBNUM( code );
        switch (itemnum) {
                case 0:         /* New */
                        break;
                case 1:         /* Open */
                        switch (sub) {
                                case 0:
                                break;
                        }
                        break;
                case 2:         /* Close */
                                        break;
                case 3: /* Quit */
                        return( FALSE );
                        break;
        }
        return( TRUE );
}

/*
*       Process selections from the Edit menu.
*/
EditMenu(code)
USHORT code;
{
        USHORT itemnum;

        itemnum = ITEMNUM( code );
        switch( itemnum ) {
                case 0:         /* Cut */
                        /* Cut the active entry to the clipboard */
                        break;
                case 1:         /* Copy */
                        /* Copy the active entry to the clipboard */
                        break;
                case 2:  /* Paste */
                        break;
                case 3:         /* Erase */
                        /* Erase the active entry (displayed or hi-lited) */
                        break;
                case 4:         /* Undo */
                        break;
        }
        return( TRUE );
}
```

Gadgets

Gadgets are regions on the display which the user can interact with using some defined action with the mouse and the keyboard.

Intuition has some pre-defined gadgets, such as the sizing gadget, and you can design your own.

There are 4 basic types of gadgets:

Boolean - these elicit yes/no type answers from the user.

Proportional - these can retrieve a proportional setting from the user or convey proportional information.

String - get text from the user.

Integer - a string gadget which gets an integer value only.

An application can receive events that tell when a gadget has been selected by the mouse (GADGETDOWN) and when the user has confirmed the gadget by releasing the mouse over the gadget (GADGETUP). To elicit this behavior, set the flags GADGIMMEDIATE and RELVERIFY.

If only the flag GADGIMMEDIATE is set, a mouse down over a gadget will immediately report gadget selection.

If neither GADGIMMEDIATE or RELVERIFY are set, you will get only mouse movements.

You can combine many gadgets in one Window to form a complex input device such as a scrollbar with arrows.

Gadgets in Detail

Gadgets can be added to a Window by either linking them into the Window->FirstGadget field before the OpenWindow() call is made or by calling one of the following two routines:

> **AddGadget(window, gadget, position)** - add a gadget to the window. position specifies the position in the list of gadgets.
>
> **AddGList(window, gadget, position, numgad, requester)** - add a list of (numgad) gadgets to a window or requester.

Their converse routines to remove Gadgets from an active window are:

> **RemoveGadget(window, gadget)** - remove a gadget from a window.
>
> **RemoveGList(window, gadget, numgad)** - remove a specified number of gadgets from a window.

RefreshGadgets(gadgets, window, requester) - redraw the gadgets in a window.

RefreshGList(gadgets, window, requester, numgad) - redraw a specified number of gadgets.

ActivateGadget(gadget, window, request) - activate this string gadget. If the gadget lives in a requester, request should point to the requester.

OnGadget() - enable this gadget for selection by the user.

OffGadget() - disable input from this gadget. It will be "ghosted".

If you wish to change a gadgets state other than On/Off, you must Remove it, change it, Add it and Refresh it.

Gadget Events

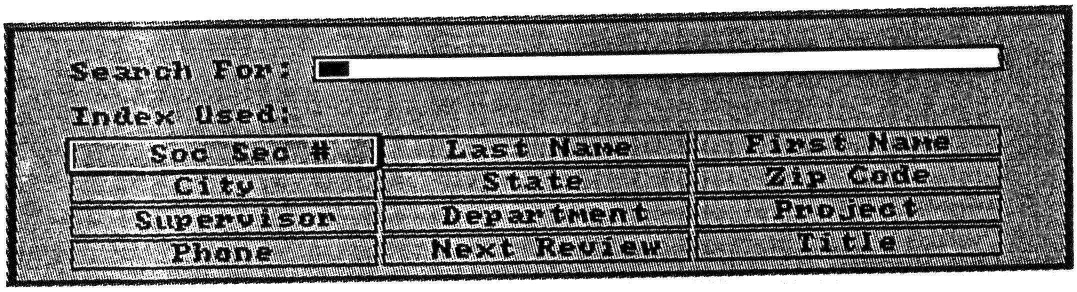
```
struct IntuiText MyText = {
        2, 1, JAM1, 30, 2, NULL, (UBYTE  *) "Gadget", NULL
};

struct Gadget MyGadget = {
        NULL,                       /* Next Gadget */
        10, 10,                     /* Left Edge, Top Edge */
        50, 12,                     /* Width, Height */
        GADGHCOMP,                  /* Highliting Flags */
        GADGIMMEDIATE | RELVERIFY,/* Activation Flags */
        BOOLGADGET,                 /* Gadget Type */
        NULL,                       /* ItemFill */
        NULL,                       /* SelectFill */
        &MyText,                    /*  Gadget Text */
        0,                          /* Mutual Exclude Bits */
        NULL,                       /* SpecialInfo */
        0x01,                       /*  GadgetID */
        NULL                        /* User Data */
};

/*
*       Our main Intuition state loop:
*       Call thusly: while (!GetIntuiMessages()) { Wait(... }
*/
int GetIntuiMessages()
{
        register struct IntuiMessage *m;
        register ULONG  Class = 0;
        register struct Gadget *G;
        register short  Code;
        short mousex, mousey;

top:    if (m = GetMsg(w->UserPort)) {
                Class = m->Class;
                Code  = m->Code;
                G     = (struct Gadget *) m->IAddress;
                mousex = m->MouseX;
                mousey = m->MouseY;
                ReplyMsg(m);
        } else return(0);

        if (Class == GADGETDOWN) {              /* Do nothing for now */
        }
        else if (Class == GADGETUP)  {
                switch(Code) {
                        case 0x01:
                                DisplayBeep(NULL);  /*  Beep the display */
                        break;
                }
        }
        goto top;
}
```

Requesters

Requesters are like Menus except they Require a response from the user.

A Requester must contain at least one gadget to elicit the required response and terminate the Requester.

A Requester may be brought up under program control directly using Request() or by a double mouse click if set up using SetDMRequest()/ClearDMRequest().


Search For:
Index Used:
Soc Sec #
Last Name
First Name
City
State
Zip Code
Supervisor
Department
Project
Phone
Next Review
Title

This Requester is composed of one string gadget and many boolean gadgets which are TOGGLESELECT. Mutual exclusion is handled by the application.

A Requester is created by calling InitRequester() and then filling in the specific fields of the Requester data structure.

A Requester is removed if a user selects a gadget that is defined with the flag ENDGADGET in the gadget->Activation flags or by the application explicitly calling EndRequest(requester, window).

Using Requesters

```
#include <exec/types.h>
#include <intuition/intuition.h>

extern ULONG WindowBits;

#define       OK_GADGET     0x01
#define       OTHER_GADGET  0x02
#define       ALEFT_EDGE    169
#define       ATOP_EDGE     45
#define       AWIDTH        320
#define       AHEIGHT       62

struct IntuiText info_text = {
       2 , 1 , JAM1 , 16, 10, NULL , (UBYTE *) "Do You Want To Continue", NULL
};

struct IntuiText ok_it = {
       2 , 1 , JAM1 , 16, 1, NULL , (UBYTE *) "OK", NULL
};

static unsigned short button_border_pairs[] = {
       -1, -1, 51, 0, 51, 11, 0, 11, 0, 0
};

static unsigned short info_border_pairs[] = {
       2, 2, 318, 2, 318, 59, 2, 59, 2, 2, 317, 2, 317, 59,
       3, 59, 3, 2, 316, 2, 316, 59, 4, 59, 4, 2
};

static struct Border button_border = {
       0, 0, 2, 2, JAM1, 5, (short *) button_border_pairs, NULL
};

static struct Border info_requestor_border = {
       0, 0, 2, 2, JAM1 , 13, (short *) info_border_pairs, NULL
};

static struct Gadget ok_gadget = {
         &other_gadget , -71 , -20 , 50 , 10 ,
         GADGHCOMP | GRELRIGHT | GRELBOTTOM ,
         RELVERIFY | ENDGADGET , BOOLGADGET | REQGADGET ,
         (APTR) &button_border , NULL , &ok_it , NULL , NULL ,
         OK_GADGET , NULL
};

struct Requester info_requester;
```

Using Requesters (cont'd.)

```
        if (!Request(&info_requester, w)) return;
        while (!stopflag) {
                while (!(message = GetMsg(w->UserPort)))
                        Wait (WindowBits);
                class = message->Class;
                ret_gadget = (struct Gadget *) message->IAddress;
                ReplyMsg(message);
                if (class == GADGETUP)
                        switch (ret_gadget->GadgetID) {
                                case OTHER_GADGET:
                                        stopflag = TRUE;
                                        break;
                        }
                if (class == REQCLEAR) stopflag = TRUE;
        }
}
```

Keyboard Processing

There are two ways your application can receive input from the keyboard:

Open up the console device yourself and attach it to your window.

Let Intuition manage keyboard events for you.

If you let Intuition manage the console for you, you can choose to receive either VANIILLAKEY or RAWKEY events.

VANILLAKEY will return an actual ASCII value based on a very simple, fixed mapping of the keyboard. You will not be able to receive information about "special" key sequences.

RAWKEY will return a raw key code corresponding directly to the physical key location on the keyboard. It is up to your application to process the keystrokes as they come in. Combining this flag with MOUSEBUTTONS and MOUSEMOVE essentially eliminates the need for opening the console device to receive user input.

Since each national Amiga keyboard has keycaps in different places, the RAWKEY numbers for each correspond to different ASCII characters. Therefore, if you use RAWKEY, you must use the console.device function RawKeyConvert() to convert the raw keys to ASCII based on the system national keymap.

Keyboard Processing Under Intuition

```
#include <intuition/intuition.h>

long KeyPressed = 0;
UBYTE kbuffer[16];

long DeadKeyConvert(msg, kbuffer, kbsize, kmap)
struct IntuiMessage *msg;
UBYTE *kbuffer;
int kbsize;
struct KeyMap *kmap;
{
        static struct InputEvent ievent = {NULL, IECLASS_RAWKEY, 0L, 0L, 0L};

        if(msg->Code & IECODE_UP_PREFIX)
                return(0);

        ievent.ie_Code = msg->Code;
        ievent.ie_Qualifier = msg->Qualifier;

        ievent.ie_position.ie_addr = *((APTR *) msg->IAddress);
        return (RawKeyConvert(&ievent, kbuffer, kbsize, kmap));
}

/*
* Our Intuition event loop. Shortened for our example.
*/
int GetIntuiMessages()
{
        register struct IntuiMessage *m;
        register ULONG  Class = 0;
        register short  Code;
        long temp;

top:    if (m = (struct IntuiMessage *)GetMsg(w->UserPort)) {
                Class = m->Class;
                Code  = m->Code;
        } else return(0);

        if (Class == RAWKEY) {
                switch (Code) {
                case 0x41:   /* Backspace */
                        break;
```

Keyboard and Intuition (cont'd.)

```
        case 0x44:   /* Return */
               break;
        case CURSORUP:
               break;
        case CURSORDOWN:
               break;
        case CURSORRIGHT:
               break;
        case CURSORLEFT:
               break;
        case 0x5f:   /* Help */
               break;
        default:
               strncpy(&kbuffer, '0', 15);
               temp = DeadKeyConvert(m, &kbuffer, 15L, NULL);

               if (temp > 0) {
                      Text(RPort, kbuffer, (long) temp);
                      currentx = RPort->cp_x;
               }
        }
        ReplyMsg(m);
}
```

Note that RawKeyConvert() requires that the console device be opened. If you do not intend to use the console device for other purposes, you may open it in the following fashion:

```
struct IOStdReq    conrequest;
struct Device      *ConsoleDevice;

OpenDevice("console.device", -1L, &conrequest, 0L);
ConsoleDevice = conrequest.io_Device;
```

RawKeyConvert() specifically needs the symbol "ConsoleDevice" to be present.

AmigaDOS

AmigaDOS is not *the* Operating System of the Amiga!

Based on TriPos, the Message Passing OS from Cambridge

AmigaDOS provides and manages I/O Streams.

AmigaDOS manages filing systems and devices. It does have it's own notion of a Process, and it is based on the Exec Task

AmigaDOS provides a Command Line Interface to the User from which programs can be launched.

Some of AmigaDOS's File I/O functions. The parameters "file", "lock" and "segment" are BPTRs, "buffer" is a char * and "FileInfoBlock" is a struct FileInfoBlock:

file = Open(name, mode) - Open a file for input or output

Close(file) - Close an open file

Seek(file, position, mode) - move to a logical position in a file. The seek does not take place until an actual Read() or Write().

actual = Read(file, buffer, length)

actual = Write(file, buffer, length)

Input() - used to determine a process's initial input file handle (similiar to stdin).

Output() - used to determine a process's initial output file handle (similiar to stdout).

lock = Lock(name, mode) - attempt to lock a directory or file.

AmigaDOS (cont'd.)

UnLock(lock) - remove an existing lock.

DupLock(lock) - duplicate an existing shared lock.

Examine(lock, FileInfoBlock) - examine a file or directory associated with a lock. The FileInfoBlock is filled in with detailed information on the file or directory .

ExNext(lock, FileInfoBlock) - examine the next entry in a directory. You have to call examine the directory FileInfoBlock first. The FileInfoBlock is then modified by this call so subsequent calls will yield the next entry in the directory. Will return ERROR_NO_MORE_ENTRIES after the last entry.

Execute(commandstring, input, output) - will attempt to execute a CLI command just as if it were typed in by the user.

LoadSeg(name) - will attempt to load a load module produced by the linker into memory. The module will be scatter loaded, chaining the various segments together. You can then call CreateProc() to start executing your code.

UnLoadSeg(segment) - unload a segment previously loaded by LoadSeg().

CreateProc(name, pri, segment, stacksize) - creates a process. It allocates a process data structure from free memory, initializes it and then starts the process (which is based on an Exec Task).

Note that the FileInfoBlock structure MUST be longword aligned.

AmigaDOS - File I/O

```
#include <exec/exec.h>
#include <libraries/dos.h>
#include <proto/exec.h>
#include <proto/dos.h>

main(argc, argv)
int argc;
char *argv[];
{
	BPTR  OldDirectory, NewDirectory;

	/* Trying to get the lock is a quick way to see if the
	   directory exists and is accessable by us */
	NewDirectory = Lock("T:", ACCESS_READ);

	/* Save the directory we started in so we can restore it later */
	OldDirectory = (struct FileLock *) CurrentDir(NewDirectory);

	WriteData("dostest");

	(void) CurrentDir(OldDirectory); /* change back to original dir */
	UnLock(NewDirectory);
}

/* Open a file - If the file already exists, the Open will fail.*/
WriteData(iname)
char *iname;
{
	BPTR  ifh;

	if ((ifh = Open(iname, MODE_NEWFILE)) != 0) {
		/* Do whatever you wish to the file */
		Close(ifh);		/* Close file ASAP */
	}
}
```

This code fragment will change the current working directory, and call a subroutine that Opens a file, Reads in it's contents and Closes the file. The code then changes the directory back to the original directory.

So, we go from wherever we start out to "T:" for read access, read our file, and then go back to where we started from.

File Access Modes available: **ACCESS_READ, ACCESS_WRITE, EXCLUSIVE_LOCK, SHARED_LOCK**.

WorkBench and CLI Startup

The following code fragment demonstrates the proper method of determining whether or not you were launched from a CLI or the WorkBench and parsing various arguments from either environment.

```
#include <exec/exec.h>
#include <intuition/intuition.h>
#include <libraries/dos.h>
#include <workbench/workbench.h>
#include <workbench/icon.h>
#include <workbench/startup.h>
#include <proto/exec.h>
#include <proto/dos.h>
#include <proto/intuition.h>
#include <proto/icon.h>

extern struct IntuitionBase *IntuitionBase; /* We open Intuition elsewhere */

void main(argc, argv)
int argc;
char *argv[];
{
      struct WBStartup   *WBenchMsg;
      struct WBArg       *Arg;
      char               *s;
      BPTR               OldDirectory;

      if (!InitLibs()) cleanexit(); /* Open whatever libraries we need */

      if (argc == 0) {         /* We were launched from WorkBench */
            /* The old method was to just declare:
             * extern struct WBStartup          *WBenchMsg;
             * since the compiler startup took care of it. But it was
             * not residentable
             */
            WBenchMsg = (struct WBStartup *) argv;
            Arg = WBenchMsg->sm_ArgList;
            if (Arg->wa_Name ) {
                  if (WBenchMsg->sm_NumArgs == 2)      {
                                    /* We were launched from a Project */
                        Arg++;      /* Examine the project icon */
                  }
                  if (Arg->wa_Lock) {
                        OldDirectory = (struct FileLock *)
                                          CurrentDir(Arg->wa_Lock);
                        intensity = checkToolTypes(Arg);
```

WorkBench/CLI Startup (cont'd.)

```
            (void) CurrentDir(OldDirectory);
            }
        }
    } else {            /* We were launched from the CLI */
        /* No args */
        if (argc == 1) {
            kprintf("no args");
        }
        while (--argc > 0 && (*++argv) [0] == '-')
            for (s = argv[0]+1; *s != '0'; s++)
            switch (*s) {
                case 'd':
                    intensity = atoi(argv[1]);
                    *++argv;
                    break;
                case 'i':
                    /* GetImage(argv[1]); */
                    *++argv;
                    break;
                case 'c':
                    docolors = TRUE;
                    break;
                default:
                    kprintf("illegal option %c", *s);
                    argc = 0;
                    cleanexit();
                    break;
                }
        if (argc > 2) {
            kprintf("%s", usage);   /* Could use AutoRequest */
            cleanexit();
        }
    }
}
```

Using Icon ToolTypes

```
/*
* Open and examine the icon of the IFF file that started us.
*     Search for a ToolType that looks like: "PATTERN=number"
*     and one that looks like: "USECOLORS=YES"
*/
int checkToolTypes(wbArg)
struct WBArg *wbArg;
{
      char  **toolArray, *string;
      structDiskObject *diskObj = NULL;
      register intpat = 0;

      if (IconBase = OpenLibrary("icon.library", 33L)) {
            if (diskObj = (struct DiskObject *)
                        GetDiskObject(wbArg->wa_Name)) {
                  toolArray = diskObj->do_ToolTypes;
                  if (string = FindToolType(toolArray, "USECOLORS")) {
                        pat = MatchToolValue(string, "YES");
                        if (pat) docolors = TRUE;
                  }
                  pat = 0;
                  if (string = FindToolType(toolArray, "PATTERN")) {
                        pat = atoi(string);
                  }
                  FreeDiskObject(diskObj);
            }
            CloseLibrary(IconBase);
      }
      return (pat);
}
```

This example looked for the Icon ToolTypes:

PATTERN=number
USECOLORS=YES

Topics For Further Study

gameport

graphics - playfields, scrolling, sprites

2.0 graphics.library:
- rendering - Read/WritePixelArray(), Read/WritePixelLine()
- font - TextExtent(), TextFit()
- support - BitMapScale(), NextDisplayInfo(), OpenMonitor()

2.0 intuition.library:
- gadtools.library
- boopsi
- public screens
- window manipulation - ZipWindow(), WindowLimits()

2.0 dos.library:
- packet interfaces
- argument parsing
- notification
- file manipulation - ChangeMode(), ExamineFH()
- links
- buffered file I/O
- record locking
- environment variables

commodities exchange

2.0 workbench.library

When possible, use 2.0 features!

References Used To Prepare This Course

1. "Amiga ROM Kernel Reference Manual: Libraries and Devices", Commodore-Amiga, Inc., Addison Wesley
2. "Amiga ROM Kernel Reference Manual: Includes and Autodocs", Commodore-Amiga,Inc., Addison Wesley
3. "Amiga Hardware Reference Manual", Commodore-Amiga, Inc., Addison Wesley
4. Autodocs, Commodore-Amiga Inc.
5. Amiga Developer Conference Notes, 1988-1990, Commodore-Amiga Inc.
6. Manx Aztec C Library Source, Manx Software Systems Inc.
7. Motorola M68000 Microprocessor Reference Manual, Motorola Microsystems Inc.
8. The Fred Fish Public Domain Disk Library, Fred Fish

The Autodocs and Developer Conference Notes can be purchased directly from Commodore-Amiga:

Commodore-Amiga Technical Support
1200 Wilson Dr.
West Chester, PA 19380
1-215-431-9180

Manx Aztec C
Manx Software
1 Industrial Way
Eatontown, NJ 07724
1-908-545-2121

SAS/C
SAS Institute, Inc.
SAS Campus Drive.
Cary, NC 27513
1-919-677-8009

Fish Disks
Fred Fish
1346 W. 10th Pl
Tempe, AZ 85281

The ROM Kernel manuals can be purchased at most B. Dalton's bookstores or at your local Amiga dealer (support your local dealers!!!)

Also recommended:

The AmigaDOS Manual
Bantam Books, Commodore-Amiga

The Kickstart Guide To The Amiga
Ariadne Software

Programmers Guide To The Amiga
Rob Peck
Sybex Publishing

Fundamentals of Interactive Computer Graphics
J. D. Foley, A. Van Dam

Graphics in Overlapping Bitmap Layers
Rob Pike
ACM Transactions on Graphics 1983

Your Local User Group - Support It!

Course Evaluation

Date __/__/__

Student Name: (Optional) ________________________

Course Content

How well did the content meet the objectives of the course? 1 2 3 4 5 6 7 8 9 10
Not Very Very Well

Please list topics that you felt were:
a) presented in too much detail b) presented in too little detail
c) presented but did not apply d) not presented but should have been

Topic:	Comment
____________________	a b c d
____________________	a b c d
____________________	a b c d
____________________	a b c d

Were the course materials easy to read and follow? __Yes __ No __ Undecided

Please rate your overall satisfaction with the content 1 2 3 4 5 6 7 8 9 10
Least Most

Instructor

Did the instructor use examples that reinforced the course material? __ Yes __ No __ Undecided

Did the instructor summarize the material to emphasize the important points? __ Yes __ No __ Undecided

Did the instructor demonstrate a good understanding of the course content? __ Yes __ No __ Undecided

Please provide any other comments you may have about the instructor: ____________________

Please rate your overall satisfaction with the instructor 1 2 3 4 5 6 7 8 9 10
Least Most

Would you recommend this course to an associate? __ Yes __ No __ Undecided

Are there other topics you would like to see addressed in this format?

Thank you for taking the time to complete this survey. Your opinions are valuable and will help me to improve the quality of the material I present.. Please place any additional comments you may have on the reverse side of this sheet.

Introducing CDTV Interactive Multimedia

by David Rosen, Commodore International

Commodore has undertaken a bold initiative -- to bring CD-ROM to the consumer market. Up to now, CD-ROM has been defined as a mass data storage medium for electronic publishing accessed through a disc-drive peripheral to a desk-top computer. With CDTV, Commodore launches a new consumer electronics category, Interactive Multimedia. Commodore's goal is to transform CD-ROM from a somewhat staid professional medium into a mass, popular medium analogous to trade-book publishing, home video, video games and prerecorded audio.

In order to make CD-ROM a mass publishing medium, the computer must become a truly "user friendly" family product. It is estimated that 75 percent of U.S. households own at least one VCR while only about 25 percent have a home computer. Why?; or perhapsa better question: why has the computer remained a school- or workplace-experienced product? Through extensive research in the U.S. and Europe it has become clear to Commodore (as well as many other companies) that consumers -- including many using computers at their workplaces -- do not consider the computer a "fun" product; rather, it is often considered intimidating, surely not designed for the family- or living-room. To try to overcome this deep-seated consumer resistance, Commodore has adopted a Trojan horse strategy: the computer has been reconceived and repositioned into a multifunctional home appliance.

The CDTV player integrates an Amiga 500 computer motherboard (built on a Motorola 68000 chip) with an ISO-9660 compatible CD-ROM disc drive within a sleek, black box, a consumer-familiar form factor resembling a conventional CD-audio player or VCR. Clearly, it is designed to be part of the living or family-room, easily integrated into a household entertainment rack system. With four video-out ports, the player hooks up to any standard TV set (i.e., composite, S-Video or RF modulated) or RGB monitor. Its two RCA-type audio ports ensure compatibility with all home stereos; the player plays all CD-audio titles as well as the record industry's newest formats, CD+Graphics [CD+G] and CD+MIDI [Musical Instrument Digital Interface] for synthesizers. The CDTV player's suggested retail price is US $999. To add to its functionality, a host of accessories, including a trackball controller and mouse, are being introduced.

Changing the form factor was only the first of a number of critical decisions Commodore made creating CDTV. Equally important, Commodore replaced the traditional keyboard with a handheld remote control. The CDTV remote combines the functionality of a conventional remote with the playability of a video game device. While the CDTV remote is obviously limited for word processing or spreadsheet applications, it's appeal is in its inherent familiarity to other living-room interface devices and overall ease of use. Needless to say, when you shift the user interface from an arms-length distance of a computer monitor to a 6-to-10-foot distance of a home TV set and replace the keyboard with a remote control, screen images -- and particularly text -- must also change. Successfully redesigning on-screen graphics is one of the challenges facing CD-ROM title developers seeking to reach the consumer market.

While CDTV is being targeted to the consumer market, it is equally appropriate for the education and library as well as vertical commercial markets. With parallel and serial ports for printer and modem, with ports for a wired keyboard or floppy/hard disk drive, the base case CDTV player can be easily reconfigured into a full personal computer; it also comes with a RAM/ROM front port for a personal memory card with up to 256K storage to further enhance player functionality. Rear slots allow for easy integration of hard disk, SCSI, LAN or other specialized videocard. The use of a genlock easily integrates CDTV with a laserdisc player to create a third-level, interactive, full-motion solution. CDTV is a high-quality, affordable, all-in-one computer/CD-ROM player. Commodore is working with a host of training, catalog,point-of-sale/information, other business groups and value added resellers to develop appropriate multimedia solutions to theirspecialized needs.

However, it is in the consumer or mass market that Commodore expects to see CDTV's greatest impact. Key to success is providinga diverse selection of titles or applications. The first CDTVcatalog lists 92 titles in five categories: Arts & Leisure, Education, Entertainment, Music and Reference; Periodicals and Productivity are forthcoming. (See Table 1 for a selected sample.) The wide assortment of titles are designed for the whole family, both children and adults. However, Commodore has made a commitment to ensure a strong representation of education, learning and reference titles. Approximately half of all current and planned titles are learning related; the other half are games, simulations and entertainment oriented. This balance is appropriate for a TV-based experience as well as the challenging decade that lies ahead.

Today, there are over 400 CDTV licensees worldwide. They aredrawn from a diverse assortment of backgrounds, including many ofthe world's leading media conglomerates (e.g., Disney Software, Grolier, Lucasfilm), specialized "multimedia" publishers (e.g., Applied Optical, Discis, ICOM Simulations and Xiphias), leading games developers (e.g., Accolade, MirrorSoft and Spectrum Holobyte) and even "garage-shop" startups that are drawn to CDTV because ofthe entrepreneurial opportunities this exciting new medium makes possible. Publishing for CDTV is relatively easy and remarkably affordable, be it for an encyclopedia, a Jack Nichlaus golf game or even a way-out cyberpunk comic book.

In conclusion, CDTV is a bold initiative to bring CD-ROM tothe mass consumer market. First, CDTV represents the historic conversion of consumer electronics and computing. These two essentially parallel industries are now coming together to create not only a new type of home appliance (i.e., one with a CPU as the active component of a consumer product), but also a new generation of CD-ROM applications or titles that will provide consumers witha richer, more rewarding participatory experience. Second, CDTV addresses consumers' deep-seated resistance to home computing by establishing a new category that goes beyond the limitations of both consumer electronics (i.e., single-function products) and computing (e.g., word processing or spreadsheets).

Third, the CDTV "experience" --'(like TV viewing, VCR or laserdisc usage and videogame playing -- is based on an enhanced interactive relationship between the user and the home TV set. CDTV expands the current "interactivity experience" of TV usage: while TV viewing is (at best) reflective, VCR or laserdisc usage offers single-function involvement and videogame playing maximizes eye-and-hand coordination, the CDTV experience is participatory. Combining the multi-functionality and control of computing with the enormous storage capacity of CD-ROM, CDTV adds a new dimension -- offers an enhanced experience -- not available from any other home entertainment and information product.

CDTV Titles (selected sample)*

Arts & Leisure

Advanced Military Systems	Dominion	$39.95
Animated Coloring Book	Gold Disk	$39.95
Gardenfax (4-disc series)	CDTV Publishing	$49.95
Women in Motion (Muybridge)*	On-Line	$49.95
Our Househ	Con-Text	$49.95

Education

A Bun for Barney	Multimedia Corp.	$49.95
Barney Bear Goes to School	Free Spirit	$39.95
Cinderella	Discis	$49.9
Fun School	DataBase	$49.95
LTV English	Jeriko	$49.95
Mind Run	CDTV Publishing	$39.95
My Paint	Saddleback	$39.95
North Polar Expedition	Virgin	$49.95
Tale of Peter Rabbit	Discis	$59.95

Entertainment

All Dogs Go to Heaven	Merit	$49.95
Talking Electronic Crayon		
Battle Chess	Interplay	$59.95
Falcon	Spectrum Holobyte	$69.95
Future Wars	Interplay	$49.95
Sherlock Holmes	ICOM	$69.95
Sim City	Maxis	$49.95
Snoopy: The Case of the Missing Blanket	Edge	$49.95
Wrath of the Demon	ReadySoft	$49.95
Xenon 2: Megablast	Mirror Soft	$49.95

Music

Music Maker	CDTV Publishing	$49.95

---

*Titles subject to change without notice.

Reference

American Heritage Illustrated Encyclopedic Dictionary	Xiphias	$69.95
Complete Works of Shakespeare	Animated Pixels	$49.95
Illustrated Holy Bible	Animated Pixels	$49.95
New Basics Electronic Cookbook	Xiphias	$59.95
New Grolier Electronic Encyclopedia	CDTV Publishing	$395.00
Time Table of Science and Innovation	Xiphias	$59.95
Time Table of Business Politics and Media	Xiphias	$59.95

Finally, CDTV is a product for today and tomorrow. Its diverse titles have appeal to the whole family as the next-generation multimedia CD format. But with the ability to expand to a full computer or add a modem, printer or genlock, it gives consumers flexibility and multifunctional capabilities which they can take advantage of depending on their own needs or desires. This will become increasingly important during the 1990s as we see CD-ROM being integrated into innovative consumer media categories. Two such areas could be multimedia CD-based catalogs integrated into on-line or other telephone-network services and the creation of "multimedia homework" integrating text, audio and graphics/videodata drawn from a CD disc and other sources on videotape. This is the future. In the mean time, CDTV is available commercially in the U.S., Canada and throughout Europe.

DEVELOPING A CDTV TITLE

■ *What does it take to develop a title for CDTV multimedia?*

Because it is based on an established technology for which there is a wealth of development tools, CDTV interactive multimedia is perhaps the easiest multimedia or CD-ROM platform on which to develop. The development system and the tools are not only relatively inexpensive, but also powerful and easy to use.

First you need an idea, of course, and an implementation design. Then you choose an authoring system suitable for your design, and get going. The application is first constructed on a hard drive. From this an image is generated in the CDTV ISO-9660 format and a test disc (CD) is made. (Optionally, testing can be done from the hard drive using the CDTV emulator board, although a test disc is recommended for final testing.) The disc, hard drive or a magnetic tape is sent to a replicator for manufacturing when the testing is satisfactory.

■ *Do I have to do this all myself?*

No, there are service centers in the USA and Europe that can create the ISO image and make the test disc. Of course, you can choose any one of many replicators to manufacture your product.

■ *What if I want to use CD-quality audio?*

Then you must supply a DAT tape or CD audio disc containing your sounds or music to the replicator of your test disc. Note that the emulator board allows you to test the audio as well as the data and program execution.

■ *What development system is generally used and how much does it cost?*

An Amiga 2500/30 with a large hard drive is the recommended development platform. The Amiga was designed for multimedia use and as such is an ideal development platform for today's multimedia CD-ROM titles.

The suggested retail price of the A2500/30 is $3799 plus the cost of the hard drive. Reduced pricing for Commodore's products is typically available to members of Commodore's developer programs. The emulator is offered separately. Video and audio capture boards are available for those applications that require them.

■ *Is any special licensing required?*

There is a nominal licensing fee payable to Commodore on a quarterly basis for each bootable CDTV disc sold.

This licensing program was introduced to ensure that CDTV discs are generally uniform in approach and user interface and can be easily identified as such by the consumer. In addition, we wanted to foster a high standard of quality and reliability. The revenue generated by the program will assist in CDTV system improvements and cost reduction and help fund additional development tools and products.

■ *What do I get for my license fee?*

The fee gives you the right to use Trackbuilder software to create your master disc and use the S-Box utility to compress items on your disc. In addition, you are granted the right to use the CDTV logo in association with your title. Several routines and libraries are also available to incorporate into your title including printer drivers, fonts, a scripting language, and other useful pieces of code including CDXL.

■ *Tell me more about the authoring systems and development tools.*

Since the CDTV system is based on Amiga technology and the Amiga is the development platform, it benefits from more than five years of tool development. A multi-page directory of tools is available to the developer. In addition, the many Amiga magazines and newsletters include reviews of the latest of these tools.

Here are examples of the choices:

Authoring Systems: *AmigaVision, AMOS, The Director II, Foundation, CanDo*
Graphics: *DeluxePaint III, DigiPaint, The Art Department Professional*
Rendering: *Imagine, Sculpt 4-D, Real 3D, Caligari*
Animation: *Disney Animation Studio, DeluxePaint III*
Text Retrieval: *Xearch, Ultracard*
Video Capture: *Framegrabber video digitizer, Video Toaster*
Compression: *Squeeze Box, Time Fold*
Graphics Capture: *DigiView, Sharp JX300 and JX450 scanners with Professional ScanLab, Scanmaster, Color Pik, graphics tablets*
File Transfer: *CrossDOS, The Art Department Professional, ImageLink*
Audio Capture: *Audio Master, Perfect Sound*

What if I am working in the MS-DOS or Macintosh environment?

There are many utilities for the Amiga that allow the easy transfer of images and audio files to the CDTV format. Both systems use the ISO-9660 data format, so the extent of changes depends on the amount of code specific to the Mac or PC environment.

Generally, the driver or reader software needs to be modified. The extent of that work depends on how closely integrated the code is with the operating system or environment for which it was written.

■ How do I get answers to technical questions about CDTV multimedia?

Commodore offers a Developer Support Program which CDTV developers should join. Members receive regular mailings and access to electronic bulletin boards. If you join the Commercial Program, you are entitled to take advantage of hotline telephone support. Members of the program generally receive discounts on purchasing Commodore equipment and CDTV titles and are eligible to attend Developer Conferences.

■ What will Commodore do to help me get my titles into the market?

Commodore is a non-exclusive distributor of CDTV titles selling principally to those retailers who also stock the hardware. In the USA a phone and mail-order service is also offered.

Two opportunities are available to make purchasers aware of your title. A catalog of titles is available and is shipped with the CDTV player in English-speaking countries. A slide show of title screens with voice over in the local language describing titles is on every Welcome disc. (The Welcome disc is the multimedia "manual" shipped with the CDTV player.) In addition, a selected group of interactive demos of titles are available at point of sale and on the Welcome disc.

■ How do I join the support program?

To obtain an application for membership in Commodore's Developer Support Program, contact your local Commodore office for an application. In the USA call (215) 431-9180 or write to:

Commodore Applications and Technical Support
1200 Wilson Drive
West Chester, PA 19380

CDTV SPECIFICATIONS

Video Outputs

Analog RGB, Digital RGBI (DB-23 connector)
Composite video NTSC (RCA connector)
Component video Y-C (S connector type for S-VHS and Hi8)
RF Modulated (F connector)
Optional genlock capabilities via plug-in module: three-mode (CDTV, video source or mixed) under software control

Video Display

400 lines/Vertical frequency 60Hz
Graphic co-processor with beam synced draw, fill, and move modes (blitter)
Maximum 1MB video memory (chip memory)
Palette of 4096 colors
8 sprites per scanline

Graphics Modes

320 X 200 non-interlaced	32 colors
640 X 200 non-interlaced	16 colors
320 X 400 interlaced	32 colors
640 X 400 interlaced	16 colors

Microprocessor

Motorola 68000 16/32 bit 7.14 MHz

Custom Chips

Three Amiga-specific custom chips (Agnus, Paula and Denise) which enhance system performance by taking over tasks such as handling video, sound, direct memory access (DMA), and/or graphics

Memory

1MB chip RAM
2K non-volatile RAM reserved for system
512K ROM

Internal Slots

Intelligent video slot (for optional genlock, RF board, etc.)
DMA slot for SCSI, LAN, etc.

Specifications subject to change without notice.

CD Audio Specs

8X oversampling
Audio output : External 1.4 VRMS, 10K OHM
Frequency response : 4-20KHz
Signal/Noise : -102db
Channel Separation : -92db
Harmonic Distortion : 0.02% at 1KHz
Maximum audio capacity : 28 hours - AM quality
Sample Rates : variable from CD audio rate (44KHz) to 6KHz
Dual 16-bit D/A converter plus 64 levels of attenuation

CD-ROM Drive Specs

Sony/Philips type CD-ROM standard Mode 1 Mode 2
Data readout from disk : 153 KBytes/sec (Mode 1)
: 171 KBytes/sec (Mode 2)
Average access time : 0.5 sec
Maximum access time : 0.8 sec
Soft read error : Less than 10e-9
Hard read error : Less than 10e-12
Seek error : Less than 10e-6
Commands : CD-ROM, CD Audio, CD + G
MTBF : 10,000 P.O.H.
Standard Supported : ISO-9660
Data Capacity : 540 MB

Front Port

Stereo Headphone Jack
Port for optional personal RAM/ROM card (256K)

Rear Ports

Centronics parallel interface
RS-232 serial interface
External floppy disk drive interface (Amiga compatible)
Hardwired alternative to IR for keyboard, mouse, joystick,
2 audio output ports (RCA-type plug)
MIDI in and out

Power Consumption

50W (average AC100-240V, 50/60Hz)

Dimensions

Player : 17.25"W X 12.5"D X 3.7"H
Infrared Remote : 8.25"W X 2.75"D X .88"H

Commodore®

CDXL Overview

CDXL is a feature of the CDTV hardware design and device driver which can significantly improve the speed of the transfer of data from the disc into the player. It does not do this by speeding up the drive,but rather by permitting the data to be stored on the disc in such asway as to minimize seek time. CDXL is possible because of the unique way that the CD ROM drive is closely coupled and integrated into the Amiga elements of the player in the CDTV hardware design.

When describing CDXL it is easier to say what it is NOT than to describe what it is. Specifically, it does not use any data compression. It does not use the hardware blitter. It only uses the copper lists to open up the area into which it puts graphics. The 68000 is used to only about 8% of its capacity.

So what is it then? CDXL is a series of sophisticated ROM routines built into the CDTV firmware that allow an application to quickly locate the data it needs and move it from the disc into the player's memory in the minimum amount of time.

One of the beauties about CDTV is that it is very simple to implement from the programmer's viewpoint... and many of the other system resources are available to do other clever things.

Although we talk about CDXL in terms of motion video and demo it with motion video (1/4 to 1/3 screen HAM images at about 12 frames per second with audio playing), it can be used with ANY type of data. You can actually see when the maximum data transfer is achieved because the disc light stays on just as it does when playing music CDs.

A patent application has been filed for the techniques used in CDXL and Commodore considers the details of these techniques to be confidential. Example code and a document on using CDXL is available on a confidential basis to CDTV license holders. We think you will be impressed when you see the demos and will find many valuable uses for CDXL in your titles.

CDTV PRODUCTS EXPECTED TO SHIP IN 1991

1001	Indoor Plants	CDTV Pubs	E
1002	Garden Plants	CDTV Pubs	E
1003	Trees and Plants	CDTV Pubs	E
1004	Fruits and Veg	CDTV Pubs	E
1005	Guinness Disc of Records	CDTV Publish	E
1501	Animated Coloring Book	Gold Disc	EFIGS
1502	Advanced Military Systems	Dominion	A
1503	Women in Motion	On Line	E
1506	Family Circus Video Titling	Context	EFIGS
1508	CDTV Soccer Annual	New Media	EG
1510	Halliwell Film Guide	Beckett	E
2501	Barney Bear Goes to School	Free Spirit	A

2002	Mind Run II	CFI	EFIGSJ
2503	LTV English	Jericho	EFIGS
2502	Bun for Barney	MultiMedia	E
2505	Fun School for Under 5's	Database Educat'l	E
2507	North Polar Expedition	Virgin Mastertronic	E
2511	Paper Bag Princess	Discis	As
2512	Thomas Snowsuit	Discis	As
2513	Peter Rabbit	Discis	Asg
2514	Scary Poems	Discis	As
2515	Mud Puddle	Discis	As
2516	My Paint	Saddleback Graphics	A
2517	Mind Run	CFI	EFIGSJ
2520	Tale of Benjamin Bunny	Discis	As
2521	Moving gives me a StomachAche	Discis	As
2522	Heather Hits her First Home Run	Discis	As
2523	Cinderella	Discis	As
2524	Long Hard Day on the Ranch	Discis	As
2529	Asterix Learn French	Eurotalk	E
2530	Asterix Learn English	Eurotalk	F
2531	Asterix Learn English	Eurotalk	G
2532	Asterix Learn English	Eurotalk	I
2533	Asterix Learn English	Eurotalk	S
2534	Asterix Learn Spanish	Eurotalk	E
2535	Asterix Learn Spanish	Eurotalk	F
2536	Asterix Learn Spanish	Eurotalk	I
2537	Asterix Learn Spanish	Eurotalk	G
2538	Asterix Learn French	Eurotalk	I
2539	Asterix Learn French	Eurotalk	G
2540	Asterix Learn French	Eurotalk	F
2542	Barney Bear Goes Camping	Free Spirit	A

3001	Defender of the Crown	CDTV Publishing	A
3002	Murder, Anyone?	CDTV Publishing	A
3003	Many Roads to Murder	CDTV Publishing	A
3004	Loom	CDTV Publsihing	EFGS
3005	Indiana Jones and the Last Crusade	CDTV Publishing	EFIGS
3006	CDTV Sports Football	CDTV Publishing	A
3007	Secret of Monkey Island	CDTV Publishing	AFIG
3501	Battlechess	Interplay	A
3502	Battlestorm	Titus	EFIG
3503	Bill & Ted's Excellant Adventure	Intracorp	A
3505	Cardinal of the Kremlin	Intracorp	A
3506	Classic Board Games	Merit	AFIGSJ
3507	Flight of the Cautious Condor	Tiger Media	A
3509	Spirit of Excalibur	Mastertronic	A
3510	Falcon	Mirrorsoft	E
3512	Herewith the Clues	Domark	EFG
3513	Hound of the Baskervilles	On Line	E

3514	Jack Nichlaus Golf	Accolade	A
3517	Psycho Killer	On Line	E
3518	Sim City	Infograme/Maxis	EFIGS
3519	Trivial Pursuit	Domark	EFIGS
3522	Wrath of the Demon	ReadySoft	A
3523	Xenon 2	Mirrorsoft	EFIGS
3529	Ultimate Basketball	Context	A
3530	Ninja High School Comic Books	Wright	A
3531	Dinosaurs for Hire	Wright	A
3532	All Dogs go to Heaven	Merit	A
3533	Snoopy	The Edge	A
3534	Garfield Big Fat Hairy Deal	The Edge	E
3535	Ultimate Horseracing	Context	A
3536	Ultimate Indoor Sports	Context	A
3537	Sherlock Holmes	Icom	A
3538	Lemmings	Psygnosis	E
3540	Raffles	The Edge	E
3541	Prehistorik	Titus	EFIGS
3542	Town with No Name	On Line	E
3543	Wayne Gretsky Hockey	Bethesda	A
3546	Super Games Pak	Odessey	A
3549	Garfield Winter's Tale	The Edge	E
3550	European Space Simulator "Mega"	Coktel Vision	EFIGS
3557	Lunar Rescue	Odyssey	A
3558	Chaos in Andromeda	On Line	E

4001	Music Maker	CDTV Pubs	AEFIGS
4501	MusiColour	Virgin Mastertronic	E
4502	ReMix	Micro Deal	E
4504	Pavarotti CDTV	FIT Vision	E
4505	Christmas Karaoke	Music Sales	EFIGS
4506	Karaoke Hits I	Music Sales	E
4507	Karaoke Hits II	Music Sales	E

7001	Dr Wellman	CDTV Publishing	E
7002	Grolier's Encycl (bundle version)	CDTV Publishing	A
7003	Grolier's Encycl (deluxe version)	CDTV Publishing	A
7004	Everman's Technology	CDTV Publishing	EFIGS
7501	American Heritage Dictionary	Xiphias	A
7502	Shakespeare	Animated Pixels	E
7503	Bible	Animated Pixels	E
7504	Time Table of Business	Xiphias	A
7505	Time Table of Science	Xiphias	A
7506	World Vista Atlas	Applied Optical	A
7507	New Basics Electronic Cookbook	Xiphias	A
7508	American Vista	Applied Opti'l	A
7509	Time Table of the Arts	Xiphias	A
7514	Hutchinson Encyl'pedia	Attica	E

LEGEND:

A	AMERICAN
E	ENGLISH
F	FRENCH
I	ITALIAN
G	GERMAN
S	SPANISH
J	JAPANESE

Note: Titles and availability subject to change without notice

COMPACT DISC PLUS GRAPHICS (CD+G)

This is a partial list of audio discs that contain graphics:

ARTIST	LABEL	SELECTION #
Alphaville (Breathtaking Blue	Atlantic	81943
Branigan, Laura	Atlantic	82086
Fitzgerald, Ella	Sire	26023
Flamin' Groovies	Sire	25948
Fleetwood Mac (Behind the Mask)	Warner Bros.	26111
Harris, Emmylou (Pieces of the Sky)	Reprise	2264
Hendrix, Jimi (Smash Hits)	Reprise	2276
Information Society	Tommy Boy	25691
Isaak, Chris (Silvertone)	Warner Bros.	25156
Little Feat (Representing...)	Warner Bros.	26163
Little Feat (Hoy, Hoy)	Warner Bros.	3538
Parsons, Gram	Warner Bros.	26108
Parks, Van Dyke (Tokyo Rose)	Warner Bros.	25968
Raitt, Bonnie (Green Light)	Warner Bros.	25486
Reed, Lou (New York)	Sire	25829
Woody Guthrie Tribute	Warner Bros.	26036
Magic Flute-MIDI		

The following selections have dual inventory, i.e., they were pressed with and without graphics. Those with graphics have CD+G etched near the center hole.

Mozart/Salieri (Two One-Act Operas)	Teldec	43336
Carmina Burana	Teldec	43775
Beethoven/Liszt (9th Symphony, Piano)	Teldec	42956
Mendelssohn (3rd and 4th Symphonies)	Teldec	43676
Domingo, Placido (Belcante Domingo)	Teldec	42954
Bach (St. Matthew's Passion Highlights)	Teldec	43928
Schumann (Kinderszenen)	Teldec	43467
Mozart (Abduction from the Seraglio)	Teldec	43924
Honeymoon Suite (Racing After Midnight)	Warner Bros.	25652
Talking Heads (Naked)	Warner Bros.	25654
Anita Baker (Rapture)	Elektra	60444
Simply Red (Picture Book)	Elektra	60452
10,000 Maniacs (Blind Man's Zoo)	Elektra	60815
Phoebe Snow	Elektra	60852
Frozen Ghost (Nice Place to Visit)	Atlantic	81875
Donna Summer (Another Place and Time)	Atlantic	81987

The Amiga 3000+

System Specification

An enhanced Amiga 3000 family computer

Document Revision 0.6

1991 DevCon Release

by
Dave Haynie

July 17, 1991

Copyright © 1991 Commodore-Amiga, Inc.

IMPORTANT INFORMATION

This Document Contains Preliminary Information

The Amiga 3000+ computer described in this document is currently a work in progress. While this is an honest attempt to specify the A3000+ computer, it is still very preliminary in nature and subject to possible errors and omissions. Additionally, the specification of the A3000+ itself is still subject to possible change at this stage in the design cycle.

Commodore Technology reserves the right to correct any mistake, error, omission, or viscious lie. Corrections will be published as updates to this document, which will be released as necessary in as developer-friendly a manner as possible. Revisions will be tracked via the revision number that appears on the front cover.

This document does not constitute a guarantee that Commodore will build or offer the A3000+, as described herein or in any other form, for sale at any given time, place, cost, etc. Nor is this a guarantee that Commodore will ever release anything called "Amiga 3000+". This is simply the design and working name for a prototype next generation Amiga 3000 class computer as of the cover date.

All information herein is Copyright © 1991 by Commodore International Services Corporation, and may not be reproduced in any form without written permission.

Most of the text in Chapter 3, and the Lisa chip itself, were created by Bob Raible. Without Bob, and the other "AA" guys, Bill Thomas and Chingtao Shen, there would be no A3000+.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 Basic Description.. 1-1
1.2 Memory Organization.. 1-2

CHAPTER 2 MAIN SYSTEM CONTROLS

2.1 Gary+.. 2-1
2.1.1 Bus Timeout Support.. 2-2
2.1.2 Reset Control.. 2-3
2.1.3 Gary+ ID.. 2-3
2.2 RAMSEY.. 2-3
2.2.1 Memory Configuration.. 2-4
2.2.2 Static Column Modes.. 2-4
2.2.3 Page Mode/Static Column Detection.. 2-5
2.3 COM20020.. 2-5
2.4 DMAC.. 2-6
2.4.1 DMA and SCSI Control.. 2-6
2.4.2 Coprocessor Interface.. 2-9
2.4.3 SSPB Control.. 2-10

CHAPTER 3 THE PANDORA CHIPS

3.1 Basic Description.. 3-1

CHAPTER 4 THE DSP3210 SUBSYSTEM

4.1 WE® DSP3210 Overview.......... 4-1
4.2 DSP System Configuration.......... 4-2
4.2.1 DSP Memory Map Differences.......... 4-2
4.2.2 The DSP Reset Mechanism.......... 4-3
4.2.3 The BIO Port.......... 4-4
4.3 DSP Serial I/O.......... 4-4
4.3.1 The Phone-Line CODEC.......... 4-5
4.3.2 The Hi-Fi Audio CODEC.......... 4-7
4.4 Other DSP Information.......... 4-7

CHAPTER 5 SYSTEM EXPANSION

5.1 The Local Bus Slot.......... 5-1
5.2 The Zorro III Bus.......... 5-2
5.3 The Video Slot.......... 5-2
5.3.1 Power Connections.......... 5-2
5.3.2 Clock Signals.......... 5-3
5.3.3 Video Signals.......... 5-4
5.3.4 Audio Signals.......... 5-5
5.3.5 Port Connections.......... 5-5
5.4 The CD/AUX Connector.......... 5-6
5.5 The DSP Connector.......... 5-6
5.5.1 Power Connections.......... 5-6
5.5.2 DSP Serial Bus.......... 5-7
5.5.3 Other Signals.......... 5-7

APPENDICES

A.1 External Connector Pinouts.......... A-1
A.1.2 Mouse Port.......... A-1
A.1.2 Keyboard Port.......... A-2
A.1.3 Serial Port.......... A-2
A.1.4 Parallel Port.......... A-2
A.1.5 Floppy Port.......... A-2
A.1.6 SCSI Port.......... A-2
A.1.7 Audio Port.......... A-3

A.1.8 Video Port.......... A-3
A.2 Internal Connector Pinouts.......... A-4
A.2.1 Power Connector.......... A-4
A.2.2 CD/AUX Connector.......... A-4
A.2.3 DSP Connector.......... A-4
A.2.4 Floppy Connector.......... A-4
A.2.5 SCSI Connector.......... A-5
A.2.6 Daughterboard Slot.......... A-5
A.2.7 Video Slot.......... A-6
A.2.8 Zorro III Slot.......... A-7
A.2.9 Local Bus/Coprocessor Slot.......... A-8
A.3 System Configuration Jumpers.......... A-11
A.4 I^2C Bus Information.......... A-13
A.4.1 Configuring SSPB for I^2C protocols.......... A-13
A.4.2 A3000+ Audio Processor.......... A-14
A.4.3 A3000+ RAM/Clock.......... A-15

A.6 Additional CODEC Information.......... A-25
A.6.1 The Phone-Line CODEC.......... A-25
A.6.2 The Hi-Fi Audio CODEC.......... A-27
A.7 References.......... A-31

TABLES AND FIGURES

Figure 1-1 Amiga 3000+ System Structure 1-3
Figure 1-2 Amiga 3000+ Memory Map 1-4

Table 2-1 Gary+ Registers 2-2
Table 2-2 RAMSEY Registers 2-3
Table 2-3 Refresh Timing 2-4
Table 2-4 COM20020 Registers 2-6
Table 2-5 DMAC/SCSI Registers 2-7
Table 2-6 Control/Interrupt Bit Assignments 2-8
Table 2-7 Coprocessor Control Bit Assignments 2-9
Figure 2-1 SSPB Clock/Data Format 2-11

Table 3-1 Pandora HAM Encodings 3-3
Table 3-2 Lisa Color Mapping 3-5

Figure 4-1 DSP3210 Internal Memory Map 4-3
Table 4-1 The DSP3210 ioc Register 4-5
Table 4-2 Phone-Line CODEC Registers 4-6

CHAPTER 1

INTRODUCTION

Thou shalt not bring into the world knowledge and devices that will harm mankind.
-Herbert A. Simon

The Amiga 3000+ computer is a second generation Amiga 3000 class computer. It shares many of the Amiga 3000 system's features, while adding a significant number of enhancements, designed to increase the Amiga line's support of more advanced video and audio playback and processing. It is intended to be upward compatible with the Amiga 3000, and shares most mechanical specifications with the Amiga 3000 as well.

This document is intended to collect, in one single place, all of the hardware details of the Amiga 3000+. Obviously, it won't reach this goal completely. There is far too much information pertinent to the A3000+ to reproduce in a single document. Therefore, where appropriate, additional documents are referenced, leaving the bulk of the space available for material specific to the A3000+. This document should be all that's necessary, for example, to port a new operating system to the A3000+, assuming one obtains other generic materials, such as the microprocessor and expansion bus specifications. It also collects all A3000+ connector pinouts and I/O register definitions in a single place, though both of these items may be better described in other documents.

1.1 Basic Description

The Amiga 3000+ takes the best features of the Amiga 3000, like 32-bit architecture, video compatibility, expandability, and adds new features to better suit current and future graphics, sound, and video applications. The complete feature list is, in essence, this document. The basic highlights, though, are as follows:

- 25MHz MC68030/MC68882 Main Processor
- 50MHz ATT DSP3210 Coprocessor
- Amiga "Pandora" graphics chip set
- Improved 32-bit DRAM and SCSI DMA controllers
- Enhanced video expansion slot
- New audio I/O ports
- Support for 4MB ROM
- Built-in telephone line interface
- Built-in 2.5 Mbit/second RS-485 network port

The basic system architecture is shown in *Figure 1-1*. The organization of the A3000+ hardware is very similar to that of the A3000. The central processor is a Motorola 68030 clocked at 25MHz, with 68882 floating point coprocessor also at 25MHz. The main motherboard controller is the Gary+ chip, a custom gate array. This generates chips selects for and timing for motherboard slave resources other than Fast RAM. On the A3000+, these resources include the Amiga "Pandora" chip set (Alice, Lisa, Kelly, and Paula), up to 4MB of ROM, two 8520 Complex Interface Adaptor chips, a Standard Microsystems COM20020 network interface chip, and a custom Amiga DMA controller for SCSI bus. On the A3000+, the DMAC also contains registers for coprocessor control and the SSPB interface, a Synchronous Serial Peripheral Bus used for battery backed RAM/Clock, Audio processor, and Video Slot expansion.

To additional motherboard resources manage their own control functions. The RAMSEY chip controls up to 16MB of memory on the motherboard, with facility for burst and page-detection modes when used with static column DRAM. The RAMSEY device also acts as an address generator for the DMAC during SCSI DMA transfers. The Buster chip controls the standard Amiga expansion bus, suppoting both Zorro II and Zorro III procotols. It manages bus arbitration for both expansion, motherboard, and coprocessor slot resources.

In addition to the 68030 system, the Amiga 3000+ has an on-board DSP coprocessor, the ATT DSP3210 clocked at 50MHz. This processor has direct memory access to all 32 bit motherboard resources, just as the 68030. In addition, it has a private high speed serial bus with DMA access to 3210 internal memory. The A3000+ provides two serial bus devices, a telephone line interface CODEC and a 16 bit stereo audio CODEC. The DSP expansion connector provides two additional channels on the serial bus, to support the additional DSP peripherals.

1.2 Memory Organization

All Amigs 3000+ memory and I/O resources are memory mapped. The A3000+ memory map is illustrated in *Figure 1-2*. This is a superset of the A3000 memory map, with allocations for the new A3000+ resources and definitions for various extensions to this basic memory map to handle possible future enhancements

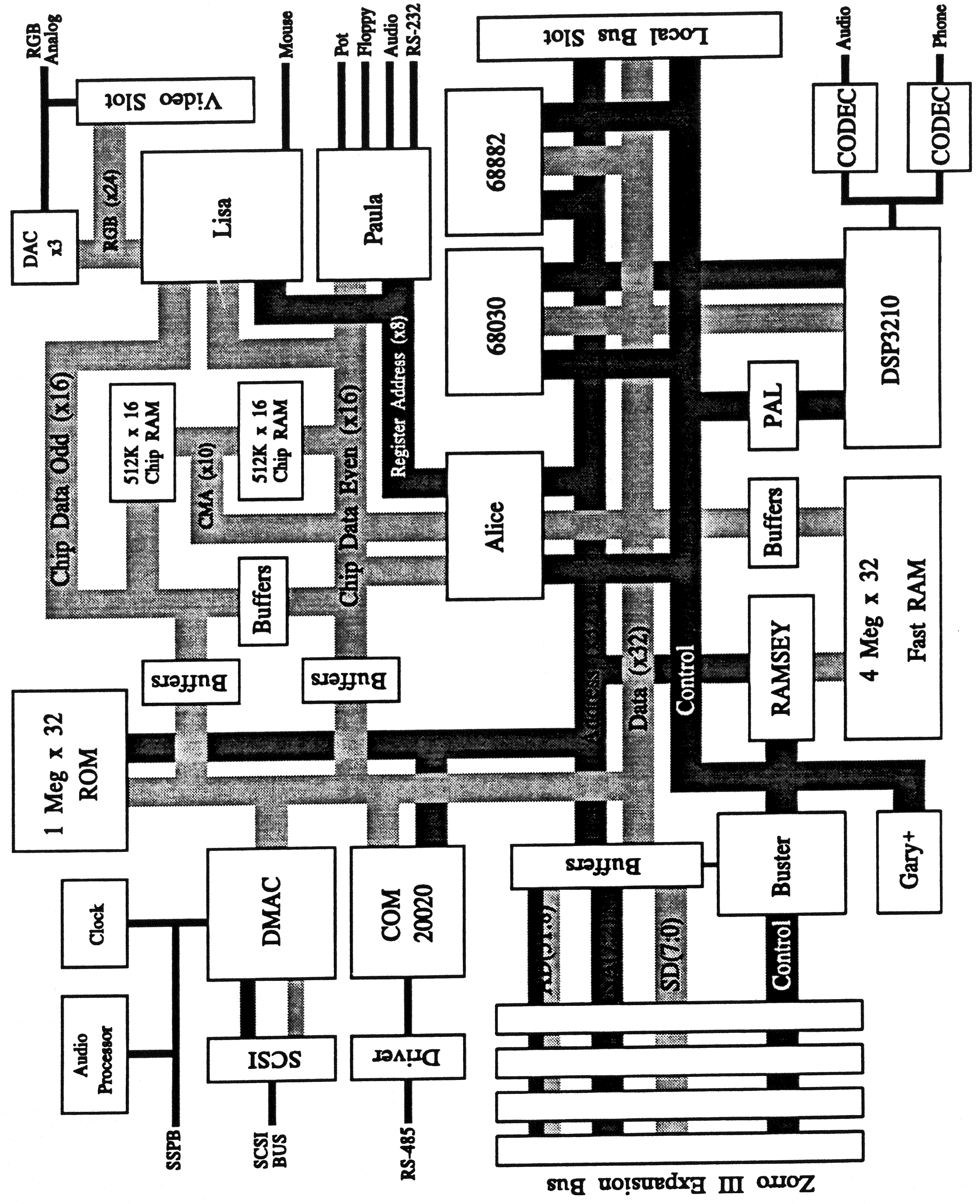

RGB Analog
Video Slot
DAC x3
RGB (x24)
Mouse
Lisa
Pot
Floppy
Audio
RS-232
Paula
Local Bus Slot
68882
68030
Audio
Phone
CODEC
CODEC
DSP3210
Chip Data Odd (x16)
512K x 16 Chip RAM
CMA (x10)
512K x 16 Chip RAM
Chip Data Even (x16)
Register Address (x8)
Alice
PAL
Buffers
Buffers
Buffers
Buffers
Address
Data (x32)
Control
RAMSEY
4 Meg x 32 Fast RAM
1 Meg x 32 ROM
Buffers
Buster
Gary+
Clock
DMAC
COM 20020
Audio Processor
SCSI
Driver
SSPB
SCSI BUS
RS-485
SD(7:0)
Control
Zorro III Expansion Bus

Figure 1-1: Amiga 3000+ System Structure

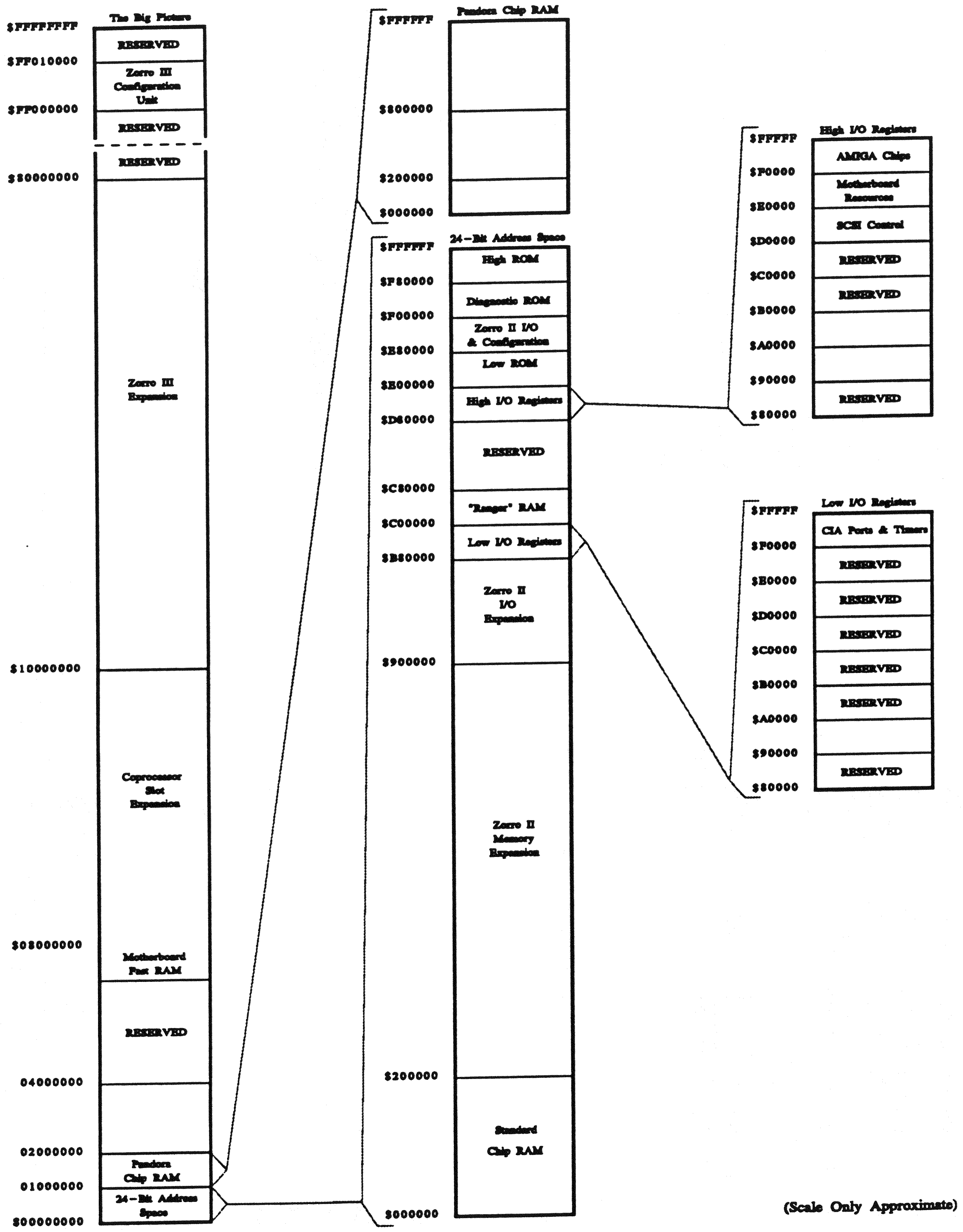

The Big Picture
$FFFFFFFF
RESERVED
$FF010000
Zorro III Configuration Unit
$FF000000
RESERVED
RESERVED
$80000000
Zorro III Expansion
$10000000
Coprocessor Slot Expansion
$08000000
Motherboard Fast RAM
RESERVED
04000000
02000000
Pandora Chip RAM
01000000
24-Bit Address Space
$00000000
Pandora Chip RAM
$FFFFFF
$800000
$200000
$000000
24-Bit Address Space
$FFFFFF
High ROM
$F80000
Diagnostic ROM
$F00000
Zorro II I/O & Configuration
$E80000
Low ROM
$E00000
High I/O Registers
$D80000
RESERVED
$C80000
"Ranger" RAM
$C00000
Low I/O Registers
$B80000
Zorro II I/O Expansion
$900000
Zorro II Memory Expansion
$200000
Standard Chip RAM
$000000
High I/O Registers
$FFFFF
AMIGA Chips
$F0000
Motherboard Resources
$E0000
SCSI Control
$D0000
RESERVED
$C0000
RESERVED
$B0000
$A0000
$90000
RESERVED
$80000
Low I/O Registers
$FFFFF
CIA Ports & Timers
$F0000
RESERVED
$E0000
RESERVED
$D0000
RESERVED
$C0000
RESERVED
$B0000
RESERVED
$A0000
$90000
RESERVED
$80000
(Scale Only Approximate)

Figure 1-2: Amiga 3000+ Memory Map

There are a few memory aliases present in this map, to support backward compatibility with the A3000 and earlier systems. Chip RAM, for example, is mapped as in earlier systems at $00000000-$001FFFFF. Since the orginal Amiga architecture supported only 2MB of Chip RAM, though, any additional Chip memory will have to be mapped non-contiguously. The A3000+ solution to this is to map the memory at $00000000 to the base of a new region, starting at $01000000. The A3000+ architecture physically supports 8MB of Chip RAM, so should an 8MB Alice be designed, the additional 6MB winds up in the $01200000-$017FFFFF range. The next 8MB has been reserved for use as Chip RAM in future systems.

Similarly, the A3000 used up all the reserved ROM space in the original Amiga memory map, the range from $00F80000-$00FFFFFF, for the 512K space used by AmigaOS 2.0. To support at least 1MB of ROM on machines with 24 bit addressing, the previously unused 512K from $00E00000-$00EFFFFF. To support additional ROM in an upward compatible fashion, the A3000+ addresses 4MB of ROM at $03C00000-$03FFFFFF. The 512K chunk at $00F80000 is also mapped at $03F80000. and the 512K chunk at $00E00000 is also mapped at $003E0000.

Some details of the A3000+ memory organization are too small to show up on this chart. The Gary+ has a number of important control registers, which are described in the next chapter. The DMAC, along with its normal DMA controller functions, also has the Coprocessor Control and SSPB registers mapped in its space. These are also covered in the next chapter. While not shown in the memory map, the DSP3210 has some internal resources that overlay the A3000+ memory as mapped by the 68030. These will be covered in the DSP3210 chapter.

CHAPTER 2

MAIN SYSTEM CONTROLS

> *We're definitely coming to the post-IBM era.*
>
> *-Joel Tessler*

There are lots of function in the Amiga 3000+, and not surprisingly, lots of control registers associated with these features. Theese control registers are physically located in the various system chips.

2.1 Gary+

The Gary+ chip is essentially the motherboard controller in the Amiga 3000+. It performs the following functions:

- Chip select and sometimes timing for Alice, the two 8520s, the DMAC, the ROM, the 68882, the COM20020, and the Local Bus Slot select.
- Chip RAM buffer and CPU byte access control.
- Genlock/internal clock multiplexing
- Reset control, including powerup detect, keyboard reset with disable, and DSP reset.
- Fast interrupt disable control.
- Bus timeout and error control.
- High order address generation for Zorro II DMA.

The chip selects should be pretty self-explanatory. The Gary+ design insures that each chips gets the length and bus size it requires. The ROM cycle is programmable via a PCB jumper to either 160ns or 280ns. Access to Chip RAM is done, as in other Amiga system, according to what the Chip RAM controller, Alice in this case, is doing. It is still possible to set up bandwidth intensive display and blitter functions with Alice, so the Chip RAM access is alway potentially variable.

Gary+ has a number of programmable control registers. These registers are located on byte boundaries, in Supervisor data space, and each contains one significant bit, which will be D_7 in the access byte. The registers are shown in *Table 2-1*.

Register	Bit	Name	Function
$00DE0000	7	TIMEOUT	Controls bus timeout
$00DE0001	7	TOENB*	Time out enable
$00DE0002	7	COLDSTART	Indicates cold powerup
$00DE1000	7	DSPRST*	DSP reset line
$00DE1001	7	KBRSTEN	Keyboard reset control
$00DE1002	7	GARYID	Serial ID code

Table 2-1: Gary+ Registers

2.1.1 Bus Timeout Support

The Gary+ timeout support is necessary to trap processor access to unmapped memory resources. In early Amigas, the motherboard controller would simply run zero wait state cycles for all memory accesses unless instruction to add waits for Chip bus or expansion resources. This policy has two problems. First of all, it makes detecting access to non-existant memory very difficult. Secondly, it pretty much requires one central agency to do all of the cycle timing in the system. Being a more sophisticated Amiga system, the A3000+ really needs the ability to trap random addressing in hardware. And cycle termination isn't central; it comes from several different places, such as Gary+ itself, RAMSEY, Buster, the 68882, and a possible slave device in the Local bus slot.

At power up, the TIMEOUT and TOENB* registers are both zero, which indicates that the default timeout is selected. Default timeout yields an automatic 32 bit asynchronous cycle termination (via /DSACK$_0$ and /DSACK$_1$) after 9µS, effectively ignoring the access. This mode is used during power up, since the OS polls certain memory locations that may not respond, and we don't want powerup to take all day. When a one is written to TIMEOUT, the detectable timeout mode is set. This causes a termination via /BERR after 250 mS. This is the normal operating mode of A3000 class systems under AmigaOS 2.0. For special purposes, timeout may be turned off altogether. This is done by writing a one to the TOENB* register. Also, since the Chip bus may lock the CPU out for extended periods of time, all timeout logic is disabled during an access to any Chip bus resource.

2.1.2 Reset Control

Gary+ has several registers associated with system reset. The COLDSTART register detets a power up condition, so that software can distinguish cold boots from warm boots. This register will be set high on power up only; once written low, it will stay low until the next power cycle. The DSPRST* line goes low on reset and stays low until a one is written to it. This allows the 68030 to get things in order before turning on the DSP3210. The KBRSTEN bit powers up low, which allows keyboard resets to generate system reset just like in any other Amiga system. If this bit is set high, however, the keyboard reset will be ignored.

2.1.3 Gary+ ID

The GARYID register is used to get the 8 bit Gary+ ID, which identifys the type of Gary and the revision level of that type. To initiate the ID code stream, a zero is written to GARYID. The bits of the Gary ID code are then read from GARY+ in eight reads of the GARYID register. The code is extracted MSB first. The first four bits of the code indicate the type of system controller, the last four bits indicate the revision level. This register is constructed in such a way that this same technique works on the Amiga 3000. The A3000's Gary returns the code $00, the Gary+ returns the code $90.

Register	Bit	Name	Function
$00DE0003	7	TEST	Used for production test
	6,5	REFRESH RATE	Timing for refresh
	4	RAMWIDTH	Set up for x4 or x1 RAM
	3	RAMSIZE	1MB or 4MB density
	2	WRAP	Enable burst wrapping
	1	BURST	Run 68030 burst cycles
	0	PAGE DETECT	Use page-detect logic
$00DE0043	7-0	VERSION	RAMSEY chip version

Table 2-2:RAMSEY Registers

2.2 RAMSEY

The A3000+ RAM controller is an improved version of the A3000 RAMSEY chip. The RAMSEY chip directly manages 16MB of 32 bit wide Fast RAM. A number of the RAMSEY features can be tuned, depending on the memory connected. The RAMSEY registers, all 8-bit registers located in Supervisor Data space, are show in *Table 2-2*. Writes to $00DE0003 don't actually take effect until the next refresh cycle. A read loop on the changed bit can be used if it necessary to wait for a change. RAMSEY controlled Fast RAM is located from $07FFFFFF building down toward $00700000, as far as the populated memory will take it.

2.2.1 Memory Configuration

The preferred A3000+ configuration is with static column DRAM in the 256Kx4 or 1MBx4 packages. RAMWIDTH defaults to one, which sets RAMSEY up for operation properly as far as the A3000+ is concerned. With RAMWIDTH set to zero, RAMSEY will drive 1MBx1 DRAM, though the A3000+ motherboard can't physically accomodate these parts. RAMSIZE is set by J852 on the A3000+ motherboard, it reads low for 256Kx4, high for 1MBx4. The RAMSIZE register can be written to, though there's generally no need for that.

DRAM Refresh is programmed by the REFRESH RATE bits, according to the timing shown in *Table 2-3*. The refresh time may have to be adjusted for the memory mode in effect. In basic access mode, the DRAM need one refresh cycle every 15.625µS or more. However, during a static column cycle resulting from page-detect mode, the DRAM need a refresh cycle every 10 µS or more. Any program modifying the OS or powerup settings of these bits needs to take this into acount.

RATE	# Clocks	Interval (µS)
00	154	6.16
01	238	9.52
10	380	15.2
11	N/A	No Refresh

Table 2-3: Refresh Timing

2.2.2 Static Column Modes

On powerup, RAMSEY runs a five clock cycle to DRAM. This cycle will work with either page mode or static column DRAM rated for 80ns operation. There are two enhanced modes that take advantage of static column memory to speed things up. The first of these is the burst mode, which is set by setting the BURST bit high. This causes RAMSEY to assert /CBREQ when accessed and run a 68030 style burst cycle when /CBACK is asserted by the current bus master. Burst cycles take two clocks each. The burst cycle operation is modified by the WRAP bit. The 68030 burst cycle always starts on a longword that the 68030 wants, and in the process gets the other three in the natural quadlongword defined by the first access. This means that burst wraps around behind the first longword when that longword isn't first in the quadlongword. With BURST set to one, natural 68030 burst takes place. With BURST set to zero, the burst cycle won't wrap backwards, but instead terminate with less than four longwords fetched. Note that the 68040 doesn't support aborted bursts, so WRAP much be one for most 68040 systems.

The other enhanced RAMSEY mode is called page-detect mode. When RAM is first accessed, RAS is held low after the cycle completes. This leaves the current RAM page "open", and the RAM bank will act like static RAM while accesses continue on this page. While accesses continue on-page, access time is driven by column rather than the longer row access time.

Memory cycles complete in three clocks while on-page. Comparators in RAMSEY monitor the new row addresses for every new cycle until a refresh comes along, which can be for up to 10µS The downside of this mechanism is that, when the page detector finds a page miss, the currently open page must be closed properly before a new one can be opened. This takes seven clocks, two to close the page, five to normally open the new one. This mode tends to work very well when memory accesses tend to be linear. It helps considerably during access by the DMAC or DSP3210, both of which have very linear access patterns. Under normal 68030 and AmigaOS use, this mode is sometimes a win, sometimes a loss.

Note that, due to a chip bug, original A3000 versions of RAMSEY (version code $0D) don't correctly support this mode. The A3000+ RAMSEY supports this mode properly. Any program messing with RAMSEY control registers should check for a version $0E or later RAMSEY before enabling this feature.

2.2.3 Page Mode/Static Column Detection

RAMSEY was originally designed to use static column memory exclusively. Because of these, there's no explicit support of page mode DRAM, though of course, page mode memory will work as long as no special features are enabled. The trick is detecting that page mode memory is in the A3000+ in the first place. The method for checking is as follows:

1) Disable all interrupts
2) Turn page detect mode on via the PAGE bit (be sure to wait until it takes effect).
3) Write $5AC35AC3, $AC35AC35, $C35AC35A, $35AC35AC to four consecutive longwords in a quadlongword.
4) Turn page detect mode off via the PAGE bit (again, wait for it to take effect).
5) Compare what's now in the selected quadlongword with what was written. If the data is correct, that bank has all static column DRAM in it.
6) Repeat steps 2 through 5 for each bank of memory.
7) Re-enable interrupts. If any bank contains page mode DRAM, none of the static column features can be used.

2.3 COM20020

The COM20020 Universal Local Area Network Controller is the engine behind the low cost RS-485 network in the A3000+. This device implements the physical layer of a 2.5 Mbps deterministic, token passing network interface. The device has 2K of on-chip dual port RAM for buffering of packets in both directions

The A3000+ configures the COM20020 in a bussed network configuration, based on the RS-485 twisted pair bus standard. The A3000+ uses RCA-type phono jacks for the physical connection. The connector is self-terminating; when a jack is plugged in, the terminator is removed from the bus. In this way, it's guaranteed that termination is placed at the end of the bus, and only at the end of the bus.

The COM20020 is fully described in the document COM20020 Universal Local Area Network Controller (ULANC), from Standard Microsystem Corporation, Component Products Division. The part is mapped in the A3000 system as a 32 bit device, located at $00D90000. Since the device has an eight bit data bus, each COM20020 register occupies only the first eight bits of each consecutive longword based location. The basic register map is given in *Table 2-4*.

Address	Read Function	Write Function
$00D90000	Status	Interrupt Mask
$00D90004	Diagnostic Status	Command
$00D90008	Address Pointer High	Address Pointer High
$00D9000C	Address Pointer Low	Address Pointer Low
$00D90010	Data	Data
$00D90014	Reserved	Reserved
$00D90018	Configuration	Configuration
$00D9001C	TentID/NodeID/Setup	TentID/NodeID/Setup

Table 2-4: COM20020 Register Mapping

2.4 DMAC

The DMAC is an improved version of the A3000 DMAC. This chip's main function is to act as a data transfer unit to the Western Digital 33C93 SCSI controller. The A3000+ version of this part has a FIFO increased to eight longwords deep, and it can now run as fast as the 68030 to 32 bit memory. It can also handle word-aligned DMA transfers, where the earlier device required longword aligned buffers. The new DMAC is designed to be compatible with the A3000 and A3000 software, so some bit assignments and features may seem strange.

Due to pin count restrictions, most DMAC control registers are not mapped strictly according to 68030 bus rules. All DMAC registers are accessed as longword-wide registers. However, they will always behave as whole longwords, even if accessed via word or byte instructions; it is impossible to independently access individual words or bytes within these registers. Therefore, any writes to DMAC registers must write all significant bits, and any reads from DMAC registers will appear to read the entire register, even if the machine op-code would claim otherwise.

2.4.1 DMA and SCSI Control

The DMAC and WD 33C93 SCSI controller chips work together as a team for most kinds of SCSI transfers. The WD 33C93 registers are actually mapped into the DMAC register space, as shown in *Figure 2-5*. Registers labelled "Read" are read-only registers, and "Write" are write-only. Those labelled "R/W" can be read or writtern. "Strobe" registers will cause a specific action to take place when read or written, regardless of the data value involved. The WD3393 registers are mapped a little strangely. There are two WD33C93 registers, each of which is a byte-wide register. These were mapped in the A3000 based on 68030 behavior, rather than 68030 specificatios, so properly designed 68040 cards could not access these registers as bytes. Allowing access to SASR_L solved this for the 68040, but due to a cache bug in the 68030, this caused a problem correctable only with the MMU. The mapping in the enhanced DMAC provides byte-wide mapping that works for 68030 or 68040.

Register	Name	Type	Function
$00DD0004	WTC	R/W	Word Transfer Count (obs)
$00DD0008	CONTR	R/W	Control Register
$00DD000C	ACR	R/W	Address Control Register
$00DD0010	ST_DMA	Strobe	Start DMA
$00DD0014	FLUSH	Strobe	Flush FIFO
$00DD0018	CLR_INT	Strobe	Clear Interrupts
$00DD001C	ISTR	Read	Interrupt Status Register
$00DD003C	SP_DMA	Strobe	Stop DMA
$00DD0040	SASR_L	Write	WD33C93 SASR long (obs)
$00DD0041	SASR_B	Read	WD33C93 SASR byte (obs)
$00DD0047	SCMD_B	R/W	WD33C93 SCMD byte
$00DD0049	SASR_B	R/W	WD33C93 SASR byte

Table 2-5: DMAC/SCSI Registers

The WTC register is considered obselete, and is not actually supported in the A3000+ DMAC. It can be used to determine if new or old style DMAC is in a system, however. Bit two of the register is fixed at zero in the new part, but is a read/write bit in the old part. So writing a longword with bit two set, then reading it back, will determine which part is installed in a system. All A3000+ systems will have the new part, though A3000 systems may have either.

The DMAC control register, CONTR, contains four significant bits in a 32-bit word, as shown in *Table 2-6*. The DMAENA bit, reads high end DMA is enabled, low when DMA is disabled. Reset or the SP_DMA strobe cause this to go low, while the ST_DMA strobe causes it to go high. The PREST bit will reset the WD33C93 if a high is written to it, while low lets the WD33C93 operate normally. The INTENA enables the generation of a system interrupt by the DMA controller. Finally, the DMADIR bit indicates the direction of data transfer for a DMA operation. A high written here causes a data transfer from Amiga memory to the SCSI bus, while a low here causes data transfer to go from the SCSI bus to Amiga memory.

The ACR indicates the starting byte in the 32 bit address space for a DMA transfer. Note that the ACR, physically contained in the RAMSEY chip, will actually round any value written to it down to an even word, since the DMAC doesn't support odd-byte aligned transfers. This register must be updated for every new DMA transfer. Note that the length of the transfer can actually be an odd number of bytes, as explained in the discussion of the FLUSH strobe.

The ST_DMA strobe starts the DMA transfer. Once the ACR and CONTR registers are properly set for a transfer, a read or write of this strobe causes the transfer to take place. The ACR must be reset for every new DMA transfer. The SP_DMA strobe causes the current DMA to unconditionally abort, and all internal DMA related registers to be reset. This location must be strobed prior to setting up a new DMA transfer. The FLUSH strobe causes the FIFO to be flushed in the case of a transfer from SCSI to main memory. This location must be strobed after the WD33C93 indicates the end of SCSI activity. If the last element of the FIFO contains one or two bytes, a word is written; if three or four bytes, a longword is written. A strobe of flush causes immediate DMA activity; the FIFO will be flushed on the instruction following the strobe, from the CPU's point of view.

Register	Bit	Name	Function
CONTR	8	DMAENA	DMA enabled
	4	PREST	WD33C93 reset
	2	INTENA	Interrupt enable
	1	DMADIR	DMA direction
ISTR	7	INT_F	Interrupt follow
	6	INT_S	Interrupt SCSI
	5	E_INT	End of process
	4	INT_P	Interrupt pending
	1	FF	FIFO Full
	0	FE	FIFO Empty

Table 2-6:Control/Interrupt Bit Assignments

The CLR_INT strobe clears all interrupts registered by ISTR, and negates the DMAC's interrupt output line. The ISTR register is responsible for recording interrupts, as shown in *Table 2-6*. Bits INT_F, INT_S, and E_INT each reflect the status of the WD33C93 interrupt line, which is an active high interrupt. The INT_P bit is low if INTENA is cleared, otherwise it indicates the status of the WD33C93 line, and will of course also indicate that an interrupt has actually be sent to the system by the DMAC. The FF bit goes high when the FIFO is full, and the FE bit goes high when the FIFO contains less than one longword.

The SASR and SCMD registers are part of the WD33C93 specification. They are fully described in the Western Digital publications "WD33C93A SCSI Bus Interface Controller" for the WD33C93A, and "WD33C93B (SBIC) Enhanced SCSI Bus Interface Controller" for the WD33C93B. The current intent is to use the latter part in the A3000+, though that depends on availability from Western Digital.

2.4.2 Coprocessor Interface

The coprocessor interface register is located at $00DD0050, with data direction register at $00DD0054. This register is used for signalling between the 68030, the DSP3210, and optionally, a Local Bus Slot device. The A3000+ system assignments for this port are given in *Table 2-7*. The 32-bit register contains eight significant bits, all of which are reset to inputs (DDR is set to $xxxxxxFF) on reset. Bits seven and six are currently unassigned.

The bit assignments start with the five and four, which are for coprocessor slot communications. The CI2P* bit is earmarked as an input from the coprocessor slot. The PI2C* bit is earmarked as an output to the coprocessor slot. These bits have no pre-defined definition, but instead are for the use of the private signalling protocol that may be desired between a coprocessor slot device and the main system.

Bit	Name	DDR	Function
7,6	N/A	I	Reserved
5	CI2P*	I	Coprocessor signal to processor
4	PI2C*	O	Processor signal to coprocessor
3	INT6*	I	DSP caused level 6 system interrupt
2	INT2*	I	DSP caused level 2 system interrupt
1	DSP1*	O	Cause DSP level 1 interrupt
0	DSP0*	O	Cause DSP level 0 interrupt

Table 2-7: Coprocessor Control Bit Assignments

The rest of the coprocessor interface register is concerned with the motherboard's DSP3210 coprocessor. The DSP3210 can generate level 6 or level 2 interrupts, in the traditional Amiga shared interrupt protocol. INT6* and INT2* should be read, respectively, by DSP interrupt handlers to determine if the DSP was in fact the device pulling either interrupt. The DSP3210 will actually turn off the interrupt it's pulling as part of the DSP to host CPU software interface protocol. The host CPU can in turn generate DSP interrupts by writing lows to either DSP1* or DSP0*. Again, software protocols are necessary to ensure that the host CPU negates either interrupt line at the appropriate time. More information on DSP and host interaction is contained in the DSP3210 chapter.

In normal use, this register should be initialized to $FF before changing any of the data direction control bits to outputs. Failure to do this may cause unhandled interrupts to either host or DSP processors. It may also cause undefined behavior in Coprocessor slot boards that take advantage of the CI2P* and PI2C* lines. An I/O reset will reset the data direction register to all inputs, but has no effect on the stored output values.

2.4.3 SSPB Control

The Synchronous Serial Peripheral Bus (SSPB) interface consists of a data register at $00DD0058 and a control register at $00DD005C. The SSPB bus is a flexible two-wire interface for control of low speed peripherals. It consists of a data line, SP, and a clock line, CNT. The SSPB control registers allow virtually any possible two-wire synchronous serial bus to be implemented under CPU control, but they provide automatic clocking with interrupt-when-done for serial protocols than can be constructed from eleven bits or less of data.

Before any SSPB activity can take place, the SSPB control register (SSPBCTL) must be set up. The bit assignments of the SSPB Control Register is given in *Table 2-8*. The XDONE bit goes high to indicate that a successful transfer has completed; it is cleared on read, a no-op on write. The INTENB bit is written low to disable interrupts, high to cause an interrupt to be generated by the DMAC when a byte has been transferred over the SSPB bus. Interrupts are generally used by all clocked SSPB protocols.

Bit	Name	Function
11	XDONE	Transfer complete
10	INTENB	Interrupt enable
9	ACNT*	Raw access to CNT line
8	ASP*	Raw access to SP line
7-0	COUNT	CNT timebase

Table 2-6: SSPB Control Bit Assignments

A high written to ACNT* will cause a low on the CNT line when the SSPB mechanism is not in a clocked transfer, and a high written to ASP* will cause a low on the SP line when the SSPB mechanism is not in a clocked transfer. ACNT* and ASP* read, respectively, the current state of the CNT and SP lines. This allows the an active programming of the CNT and SP lines, to create serial protocols not supported by the clocked SSPB protocol.

The SSPB automatic clock period is determined by the value written to the CLOCK byte. The cycle time of CNT is given by (CLOCK * 160ns), for values of CLOCK from 1 to 255. The value of CLOCK is preserved over multiple SSPB data cycles.

Once SSPBCTL has been set up, SSPB data transfer cycles may be run. The SSPB data cycle is a little unusual, in that every cycle is technically both a read and write cycle. To start an SSPB cycle, the host processor writes to the 32 bit wide SSPB data register (SSPBDAT) a data value with the lower eleven bits significant. The SSPB mechanism then proceeds to shift out this data, from bit eleven through bit zero. Bit eleven is shifted out before the first clock on CNT, and bit zero is shifted out after the last clock on CNT, as shown in *Figure 2-1*. Once the transfer is complete, the DMAC will bring the system interrupt line low, assuming the INTENB bit is set, and the host CPU will receive an interrupt, which is held until cleared by reading SSPBCTL. Once the interrupt is cleared, SSPBDAT may be read.

This event sequence holds for either reads from or writes to SSPB devices. For a write, the write data is simply written to SSPBDAT, possibly mixed in with control information, depending on the data format in use. When called in an interrupt server chain, the SSPB server reads SSPBCTL, checking for XDONE. If asserted, SSPBDAT may optionally be checked for any sort of status the protocol in use would demand. For a read, each significant data bit is set high in a write to SSPBDAT. Upon receipt of an interrupt, the interrupt server gets the XDONE bit from SSPBCTL and then the read data and possible status from SSPBDAT.

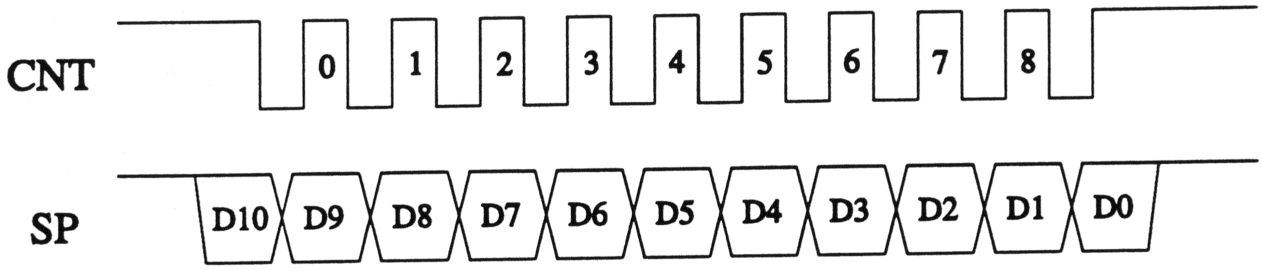


Figure 2-1: SSPB Clock/Data Format

The actual SSPB protocol used depends on what the interface is used for. At present, the A3000+ has two SSPB devices on the motherboard, the battery backed clock/RAM and the audio processor. Both of these devices follow the Phillips I^2C bus protocol. Appendix 4 contains the details on how to configure the SSPB system for I^2C compatible cycles, and also information on the clock and audio processor register maps.

CHAPTER 3

THE PANDORA CHIPS

They're something that I've never seen before
-R.E.M.

The Amiga 3000+ system is supported by a new Amiga gcaphics chip set, called the Pandora, or Advanced Amiga, chip set. This system greatly enhances the Amiga's video display capabilities, while remaining upward compatible with the Amiga ECS chip set, and retaining all of the graphics features unique to previous Amiga systems.

3.1 Basic Description

The Pandora chip set consists of three full custom LSI chips, working together as a single unit. Alice, based directly on the 2MB ECS Agnus chip, is the main Amiga chip bus controller. It is responsible for generating video and memory timing for the other chips, and it contains bimmer and copper units. While Alice is still a 16 bit chip, it now can direct 32 bit and/or double speed transactions on the Chip bus, and is simpler to interface to the A3000 style 32 bit Chip memory bus. Paula is the chip responsible for 8-bit audio output, floppy disk and RS-232 serial I/O, and potentiomenter inputs. The Paula used in the Pandora chip set is the same Paula used in classic and ECS Amiga chip sets.

Completely new for Pandora is the Lisa. Lisa is a new full custom design, a replacement for Denise. Implemented in 1.5µm CMOS, Lisa has a 32 bit Chip bus interface, and with 80ns DRAM, double fetch cycles can fetch 64 bits of data in a single Chip bus cycle, a factor of four improvement over classic and ECS video fetch cycles. The Lisa chip output is 24 bits of digital

RGB video. High quality RBG analog output from the Lisa digital video is generated by an off-the-shelf video DAC, which is currently the Analog Devices ADV7120 Triple 8-bit Video DAC. A summary of Pandora features includes:

- 32 bit wide data bus supports input of 32-bit wide bitplane data and allows doubling of memory bandwidth. Additional doubling of bandwidth can be achieved by using Fast Page Mode RAM. The same bandwidth enhancements are available for sprites. Also the maximum number of bitplanes useable in all modes is increased to eight.
- The Color Palette has been expanded to 256 colors deep and 25 bits wide (8 red, 8 green, 8 blue, 1genlock). This permits display of 256 simultaneous colors in all resolutions. A palette of 16,777,216 colors is available in all resolutions.
- 28Mhz clock input to Lisa allows for cleaner definition of HIRES and SHRES pixels. Alice's clock generator is synchronized by means of Lisa's 14MHz and SCLK outputs.
- A new register bit allows sprites to appear in the screen border regions..
- A bitplane mask field of 8 bits allows an address offset into the color palette. Two 4-bit mask fields do the same for odd and even sprites.
- In Dual Playfield modes, two 4-bitplane playfields are now possible in all resolutions.
- Two extra high-order playfield scroll bits allow seamless scrolling of up to 64 bit wide bitplanes in all resolutions. Resolution of bitplane scroll, display window, and horizontal sprite position has been improved to 35ns in all resolutions.
- A new 8 bitplane HAM mode has been created, 6 bits for colors and 2 for control bits. All HAM modes are available in all resolutions (not just LORES as before).
- A reset input pin has been added, which resets all the bits contained in registers that were new for ECS or LISA.
- Sprite resolution can be set to LORES, HIRES, or SHRES, independent of bitplane resolution.
- Attached Sprites are now available in all resolutions.
- Hardware Scan Doubling support has been added for bitplanes and sprites. This is intended to allow 15KHz screens to be intelligently displayed on a 31KHz monitor, and share the display with 31KHz screens.

While its possible to fully describe each and every display mode, this isn't really necessary. First of all, there are now an awful lot of combinations available. Since the A3000+ supports the full bandwidth Pandora Chip RAM, all display modes are possible. Basically, all display modes, interlacing, and resolution are independent of one another. Display modes can be straight bitmap, from 1 to 8 bitplanes. There is also 6 bitplane Extra HalfBright, and both 6 and 8 bitplane Hold and Modify. Resolutions are a combination of pixel speed and horizontal refresh. Pixels can be 140ns, 70ns, or 35ns. Horizontal refresh rates are usually 15.7kHz or 31kHz. Some of the combinations are given in Appendix 5.

CHAPTER 4

THE DSP3210 SUBSYSTEM

> *A live wire; barely a beginner but just watch that lady go!*
>
> *-Van Halen*

The WE® DSP3210 is a floating-point digital signal processor, extremely well suited for use as a coprocessor in a microcomputer system such as the A3000+. The DSP3210 will speed up many of the things currently done on Amigas, and provide capabilities far beyond those built into current generation Amiga systems.

4.1 WE® DSP3210 Overview

The DSP3210 is a floating-point digital signal processor, capable of running up to four memory accesses per instruction cycle, and up to 12.5 million instructions per second and 25 million floating-point operations per second. The DSP3210 architecture seeks to eliminate the problems that plagued earlier digital signal processors used as graphics, sound, and general signal coprocessors in a microcomputer environment. Rather than being limited to a few hundered bytes of on-chip RAM and a few thousand bytes of off-chip RAM, the DSP3210 couples 8K of very fast on-chip RAM with full 32-bit bus master capability, allowing it access to all 32-bit memory in the A3000+. Additionally, the AT&T Visable Caching Operating System® (VCOS) provides a realtime multitasking kernel for the DSP3210. This allows applications to use the DSP3210 as freely and as easily as they would use the host processor; within the limits of available processor time, no one application can tie up the DSP3210. Also, since VCOS supports multiple DSP3210s, even processor saturation may in time be solved with addtional expansion hardware.

4.2 DSP System Configuration

The basic DSP3210 exists as a normal motherboard bus master, with the highest DMA priority in the A3000+ system. It can run full speed 68030 compatible cycles, including burst-mode to motherboard and Zorro III expansion memory. Since the DSP is not register mapped, the communications protocol between it and the host processor is mainly an agreement in software. The system hardware provides a special DSP reset line which hold the DSP off the bus until the host processor has time to initialize the communications protocol. This reset line is controlled via a register in the Gary+ chip. Once the DSP is up and running, communcations between it and the host processor are helped by the ability to interrupt one another, via the Coprocessor Communication Register, from the host processor side (described in the System Controls chapter), and the DSP3210's BIO port., from the DSP3210 point of view.

The DSP3210 serves two main functions; that of a simple mathematics and signal processing resource, and that of an I/O processor. As a signal processing resource, applications programs can, though the Amiga-VCOS interface, run DSP3210 programs for a variety of jobs. It's also possible that software will be created to use the DSP transparently to the application; a high-level mathematics library is one example of this. Most number processing applications are scheduled under VCOS as non-realtime jobs.

For I/O processing, the DSP3210 has a high speed serial bus which can access the internal RAM via transparent DMA. The A3000+ provides a telepone line CODEC with telephone interface, and a stereo Hi-Fi audio codec with line, microphone, and headphone interfaces. Most I/O interaction, such as realtime audio and modem/FAX applications are scheduled under VCOS as realtime jobs.

4.2.1 DSP Memory Map Differences

While the DSP3210 shares the 32-bit linear address space of the A3000+ host processor, its internal resources are mapped over a small area of A3000+ memory, as shown in *Figure 4-1*. The DSP3210 can't access the memory "beneath" its on-chip resources, and the host processor can't access the DSP3210 on-chip resources. The main on-chip resources are Boot ROM (1K), on-chip I/O (1K), on-chip RAM (8K), and reserved area (54K). Note that the A3000+ system configures the DSP3210 to power up in µComputer mode, causing the on-chip resources to be located from $00000000-$0000FFFF. The alternate µProcessor mode will cause the on-chip resources to be located from $50030000-$5003FFFF.

The other main difference in memory mapping is that the DSP3210 doesn't dynamically size its bus. The A3000+ support logic allows it to communicate with both synchronous and asynchronous 32-bit resources. It cannot, however, access 8 or 16 bit devices, such as some I/O registers, all custom chip control registers, and any Zorro II device. The system hardware is designed to send the DSP3210 a bus error if it attempts access of any non-32 bit wide port. Normally, the DSP is expected to need access only to RAM in any case, and Fast RAM at that most of the time. It should offload any external I/O processing to the host processor.

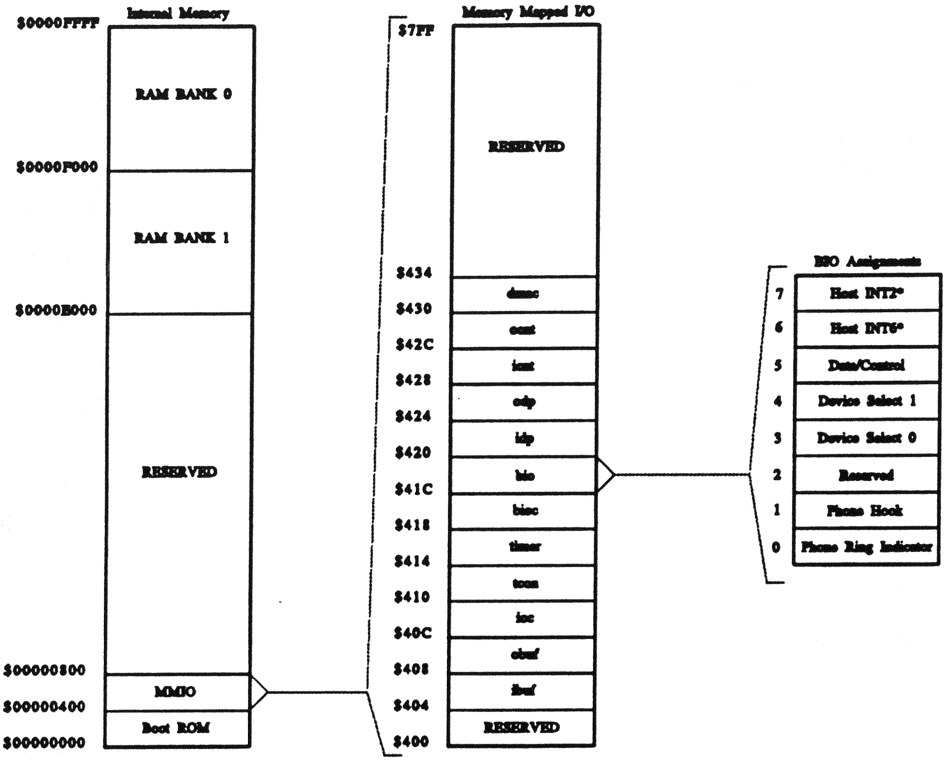

Internal Memory
$0000FFFF
RAM BANK 0
$0000F000
RAM BANK 1
$0000B000
RESERVED
$00000800
MMIO
$00000400
Boot ROM
$00000000
Memory Mapped I/O
$7FF
RESERVED
$434
dmac
$430
ocnt
$42C
icnt
$428
odp
$424
idp
$420
bio
$41C
bioc
$418
timer
$414
tcon
$410
ioc
$40C
obuf
$408
ibuf
$404
RESERVED
$400
BIO Assignments
7
Host INT2*
6
Host INT6*
5
Data/Control
4
Device Select 1
3
Device Select 0
2
Reserved
1
Phone Hook
0
Phone Ring Indicator

Figure 4-1: DSP3210 Internal Memory Map

4.2.2 The DSP Reset Mechanism

Since the DSP comes up in µComputer mode, the Boot ROM is used for reset. The A3000+ sets things up such that the DSP3210 will expect to execute code starting at $0001000. In order to let the DSP out of reset, the host processor will have to assure that this location contains the necessary DSP boot code. If this memory already contains something, it'll be necessary to first disable interrupts, save off whatever is at $00010000, move the initialization code to $00010000, enable the DSP via the Gary+ DSPRST register, wait on a signal (not an interrupt, these are disabled!) from the DSP indicating its through with the initalization code, then restore $00010000, and finally enable interrupts again. This procedure shouldn't be anywhere nearly as nasty as it sounds, and it only needs to be done once, right after reset.

Typical initialization will go something like this. The DSP wakes up and starts running code at $00010000. This code will more than likely transfer another chunk of code into the DSP's internal memory. The DSP will complete the transfer, jump to the internal code, and then signal the host processor, probably via a memory based semaphore, that it's done with the initialization code. This internal code will, at the least, allow the DSP to accept commands from the host processor. Initially, the DSP will wait for an interrupt. In a real system, interrupts will come from I/O as well as from host processor communications, of course. When the DSP gets a host attention interrupt, it'll go out to an agree-upon command buffer in external memory and figure out just what the host wants. Once it finishes the host task, it may notify the host that the task is complete, again via interrupt.

4.2.3 The BIO Port

The A3000+ system configures the DSP3210 Bit I/O (BIO) port for several system support functions. The port-line relative mapping is shown in *Figure 4-1*. The first two bits are assigned to the telephone line CODEC. The Ring Indicator bit is an input from an optoisolator in the telephone line interface, which indicates to the DSP3210 that the phone is ringing. The next bit, the Hook bit, is an output used to hang up or pick up the phone line. The bit after that, bit two, is currently reserved.

The next three bits are used to configure the serial port device under DSP3210 control. The Device 0 and Device 1 bits select one of four possible high speed serial bus devices. Device zero is the telephone line CODEC, device one is the Hi-Fi Audio CODEC, both located on the motherboard. Devices two and three are, respectively, devices on the DSP expansion port selected, respectively, by EXP0* and EXP1*. The next bit is the data/control mode toggle. Some serial bus devices have a distinction between data mode, which is the normal operating mode, and control mode, which set up various control parameters. The current serial port device powers up as the telephone line CODEC< and is selected by changing the value of the Device 0 and Device 1 bits when in command mode.

The final two bits are part of the communications protocol between the DSP3210 and the host processor. When the DSP needs to interrupt the host processor, it drives the Host INT2* bit low to generate a host bus level 2 interrupt, or the Host INT6* bit low, to generate a host bus level 6 interrupt. The interrupt server routine running on the host needs to signal the DSP3210 to clear these interrupts, otherwise the system will wind up in an infinite interrupt loop.

4.3 DSP Serial I/O

The DSP3210 has a very flexible high speed serial port, as previously mentioned. Under control of the BIO port and external logic, this serial bus manages up to four separate devices, each of which may use its own serial bus protocol. The serial port itself is controlled by the ioc register, show in *Table 4-1*.

The serial bus consists of seven interface lines. One is the frame synchronization line (SY) that's used for both input and output. There are three lines for input: clock (ICK), load strobe (ILD), and data (DI). Similarly, there are hree lines for output: clock (OCK), load strobe (OLD), and data (DO). Via the ioc register, various serial protocols can be set up. Clocks, load strobes, and the frame synchronization line can be set up either as inputs or as outputs, depending on the setting of various bits in the ioc register. The internal versus external settings are all independent of one another. When internal, the timing for ICK and OCK is set by the ICN. The AIC bit determines if ICK is internal or external, while AOC determines if OCK is internal or external. Other internal clocked timing is based on the internal clock, which is either ICK or OCK, as set in the BC field. Internal ILD is based on the internal clock, divided by 32; the AIL bit controls whether ILD is internal or external. Internal OLD can simlarly be based on the internal clock, divided by 32, or it can be run in "burst" mode, based on the status of the output buffer and the DMA controller., as set by the BO bit. Internal frame sync is based on the internal

Bit	Name	Function	Settings (0,1)
31-24	N/A	Reserved	Read as zero.
32-20	ICN	Internal clock	rate = ICN * 4.
19	OSZ	Ouput DMA size	32 bits, OLEN.
18	ISZ	Input DMA size	32 bits, ILEN.
17	OUT	Output data order	LSB first, MSB first.
16	IN	Input data order	LSB first, MSB first.
15	SAN	Sanity bit control	Sanity clear, sanity set.
14	IIC	ICK polarity	ICK true, ICK inverted.
13	BO	OLD clocked/burst	Clocked, Burst
12-10	OLEN	Ouput data length	8, 16, 32, or 24 bits long.
9	AOL	OLD source	External, internal.
8	AOC	OCK source	External, internal (ICN).
7-6	ILEN	Input data length	32, 8, or 16 bits long.
5	AIL	ILD external/internal	External, internal.
4	AIC	ICK external/internal	External, internal (ICN).
3-2	SLEN	SY frequency	Ratio = 32, 8, 16, or 32.
1	BC	SY source	ICK, OCK.
0	ASY	SY external/internal	External, internal.

Table 4-1: The DSP3210 ioc Register

clock, either 8, 16, or 32 times the period of the on-chip load signal, based on SLEN. The ASY bit sets frame sync to either internal or external operation.

The serial bus data format is also quite flexible. Both input and output data streams can be set for MSB first or LSB first. The input clock polarity can also be changed. The input data format can be 8, 16, or 32 bits long, initiated by the ILD pulse, or 32 bits long, followed by the ILD pulse, as set by the ILEN field. The output data format can be 8, 16, 32, or 24 bits long, as set by the OLEN register. Input data is read from the ibuf register, output data is written to the obuf register.

4.3.1 The Phone-Line CODEC

The first of the A3000+ serial bus devices is the Phone-Line CODEC. This is a 16-bit bidirectional ΣΔ A/D-D/A converter designed especially for use in telecommunications applications. The device maintains 80 dB S/NR and THD, and supports sample rates of 7.2kHz, 8.0kHz, and 9.6kHz. For advanced transmission protocols, such as V.32, the device contains a digital resampling interpolation filter to all resampling of the receive data at the same phase as the transmitted data. The device is sufficient to support V.33, V.32ter, V.32bis, V.32, V.29, V.27bis, V.27, V.26bis, V.22bis, V.22, Bell 212A, and Bell 103 protocols (at least some of these are supported by VCOS modules available from AT&T).

The Phone-Line CODEC works the same in either control or data modes of the serial bus, and the serial device address is zero. The DSP3210 is set up to use an external clock, with internally generated transmit load strobe, externally generated receive load strobe. The device itself contains nine addressable ixteen-bit registers, five for control purposes, four for data I/O, as show in *Table 4-6*.

All transmissions consist of a 16-bit address word and a 16-bit data word. Writes to the device are initiated by the DSP3210. The address word contains the register address as bits zero through three, zeros for the reserved bits four through fourteen, and a zero as bit fifteen, indicating a write. To read a control register, the DSP3210 sends the address word, only with bit fifteen set high, followed by a dummy data word. This doesn't immediatedly read that register, but causes a read request to be logged into the CODEC device. All reads are actually initiated by the device to the DSP3210. It sends a 16-bit address word followed by a 16-bit data word, in the same format as for write transmissions. The address word is used by the DSP3210 to figure out what is being received.

Addr	Name	Function
0	CTL0	Rate select, interpolation filter enable
1	CTL1	Rate scaling, power down, filter bypass
2	CTL2	Receive bit and baud rate selects
3	CTL3	Transmit bit and baud rate selects
4	CTL4	Receive phase adjustment
5	CTL5	Transmit phase adjustment
6	DAT0	Transmit data
7	DAT1	Interpolation filter input
8	DAT2	Receive data
9	DAT3	Interpolation filter output

Table 4-2: Phone-Line CODEC Registers

More information on the Phone-Line CODEC, such as the format of the various control registers, is available in Appendix A.6.1. Additional reference material will eventually be made available.

4.3.2 The Hi-Fi Audio CODEC

The second device on the A3000+ DSP serial bus is the Hi-Fi Audio CODEC. This device is a 16-bit bidirectional two-channel ΣΔ A/D-D/A converter. It maintains greater than 80 dB S/NR and THD, and supports a large variety of sample rates, including the audio-standard 44.1kHz and 48kHz rates. It supports programmable data formats, either signed two's complement 16-bit linear, 8-bit µ-law, or 8-bit A-law encoding. Additionally, it has a programmable input source selection for microphone or line-level inputs.

The Audio CODEC, with serial bus device address of one, has distinct control and data modes. Both modes use 64-bit data frames, transmitted LSB first. The control mode frame sets the sampling rate, data format, and various other initialization parameters, while the data mode frame is the normal I/O mechanism for D/A-A/D conversion. In control mode, the DSP3210 supplies both serial clocks, both load strobes, and the frame synchronization pulse. In Data mode, the Audio CODEC supplies both serial clocks and the frame synchronization pulse, while the DSP3210 supplies the load strobes.

The control mode frame sets a number of parameters in the Audio CODEC. It selects the data format; either mono or stereo, 16-bit twos-complement PCM linear, 8-bit µ-law companded, or 8-bit A-law companded. It sets up the sampling configuration, which includes clock source and sampling frequency. It has optional loopback bits, which allow the system to be tested, with either ADC to DAC or DAC to ADC looping. And the revision code of the CODEC is available here too, to help device drivers adjust to future versions of the device.

Data mode, as expected, contains the audio data, as defined in control mode. All frames allow programmable output attenuation on both left and right channels, from 0dB to 93dB, in 1.5dB steps, as well as full digital mute. Analog mute is also available on either output channel. Input gain can be set from 0db to 22.5dB, in increments of 1.5dB. Of course, the input source can be selected as either line or microphone. A programmable monitor mode allows the ADC output to be variably mixed with the DAC's input, in steps of 0dB (fully mixed) to an attenuation of 84dB, in steps of 6dB, as well as completely unmixed. Finally, there's an overrange indicator which indicates if either ADC channel is overdriven.

One important note is the proper protocol must be observed for switching this device between control and data modes. To go from data to control mode, it is only necessary to lower the output volume to its maximum attenuation, then bring the D/C line low, via the bio port. Next, create the desired control frame and send it, with the DCB (Data Control Handshake) bit low. Read back the control frame and examine the DCB bit. Until it reads low, continue sending the control frame and reading it back. This allows multiple Audio CODECs to exist at the same serial bus device address. Once DCB reads back low, the control frame is written back once more, only with DCB set high. Now the CODEC is ready for the new setup. The D/C line is brough high, which will put the device into data mode and also execute an offset calibration on the selected input channel.

More information on the Hi-Fi Audio CODEC, such as the detailed format of the various transmission frames, is available in Appendix A.6.2. Additional reference material will eventually be made available.

4.4 Other DSP Information

The DSP contains an on-chip DMA controller, programmed via the dmac register, counter registers icnt and ocnt, and pointer registers idp and odp. The DMA controller can be used to automate serial bus transfers. There is a timer, which is controlled by the timer register, timer, and the timer control register, tcon. The timer can be used to generate an internal interrupt. The other MMIO registers, the serial control registers ibuf, obuf, and ioc, plus the BIO port control registers, bio and bioc, have already been discussed.

The DSP3210 itself has quite a few more features. It's a full 32-bit microprocessor in its own right, with 32 bit PC, twenty-two 32-bit general purpose registers, four 40-bit floating-point assumulators, and some control registers. The processor directly supports 16 and 32-bit signed integer and 32-bit floating point numbers, and it can directly load and store 8-bit signed and unsigned and 16-bit unsigned numbers. In addition, it has functions to convert to and from floating point and 8, 16, or 32-bit integer, m-law, A-law, or single-precision IEEE format floating point formats. It also has a great deal of instructions and addressing modes to operate on the internal format data. For more information on the DSP3210, consult WE® DSP3210 Digital Signal Processor, Information Manual, Revision 1.8.1, from AT&T Microelectronics.

CHAPTER 5

SYSTEM EXPANSION

> *Have you ever wondered, about sound and vision?*
>
> *-David Bowie*

The A3000+ supports the standard forms of expansion found in all slotted Amiga computers; machine-specific Local Bus Slot, for Coprocessor devices and processor upgrades, Amiga standard expansion bus for general purpose expansion, and a Video Slot, for Genlock and other specialized video applications. In addition, the A3000+ motherboard has a few extra expansion headers to provide access to a couple of its special features.

5.1 The Local Bus Slot

The A3000+ Local Bus slot is a superset of the 200 pin Local Bus Slot used in the A3000 and A3000T computers. It is compatible with all A3000 and A3000T coprocessor devices, and adds a few new features. The full pinout of the A3000+ Local Bus Slot is given in Appendix A.2.9, while the slot is fully explained in The A3000 Local Bus Slot Specification, Revision 2.00, which is available from Commodore. The signals new to the A3000+ are as follows:

Slot Interrupt (/INT6)

This is the open-drain, shared, level six interrupt to the host CPU. The coprocessor device can generate this interrupt as long as it supplies the proper interrupt server, and has a way for that server to indentify that it was the interrupt source. Sometimes, one of the Coprocessor communications bits is used for "Interrupt generated" to the host CPU.

Coprocessor Interface (/CI2P, /PI2C)

These lines, as discussed in the Coprocessor Control Register section, are for use by the coprocessor device and the host processor as they see fit for application specific communications. The /CI2P line is earmarked as a Coprocessor Slot to host processor line, the /PI2C line for host processor to Coprocessor Slot signals.

A3000+ Detect (/SENSEA3P)

This line is attached to a 1K pullup resistor on the Coprocessor device board. If it goes high, this is an A3000 system, if low, this is an A3000+system.

5.2 The Zorro III Bus

The Zorro III bus is the standard, general purpse expansion channel on the A3000+. There are no major changes to the Zorro III on the A3000+, though all A3000+ systems are expectecd to ship with the Buster II bus controller, which implements the complete Zorro III specification. The Zorro III Bus is detailed in The Zorro III Bus Specification, Revisin 1.10, which is available separately from Commodore. The Zorro III pinout is given in Appendix A.2.8.

The one new feature supported by the A3000+ is a configurable Zorro II cache support control. Normally, 32 bit Amiga systems map Zorro II memory space geographically into cachable and uncachable. This assumption works for most devices, but can fail for shared memory coprocessors, such as the Commodore-Amiga Bridge Card. Jumper J680 is responsible for switching Zorro II mode to fully uncached.

5.3 The Video Slot

The A3000+ supports an enhanced Video Slot, which is found in-line with the first Zorro III expansion slot. This slot allows expansion access to all digital and analog video signals, clocks, and some I/O lines. This slot is mainly upward compatible with the A2000 and A3000 video slots. The A3000+ Video Slot is physically two one-piece edge card connectors, consisting of one 36 pin edge connector, called the "A" connector, and one 50 pin edge connector, called the "B" connector. The pinouts for these are given in Appendix A.2.7.

5.3.1 Power Connections

The Video Slot provides several different voltages designed to supply Video devices.

Digital Ground

Digital supply ground used by all digital devices in the A3000+.

Audio Ground

Separate ground, decoupled for use by analog audio devices only.

Main Supply (+5V)

Main digital level power supply the Video Slot. This can supply large currents. This connection is speced at 1 Amp, though with proper power budgeting can physically supply as much as 4 Amps.

Negative Supply (-5V)

Negative version of the main supply, for small current loads only; there's a total of 0.3 Amp for the entire A3000+ system.

High Voltage Supply (+12V)

Higher voltage supply, intended for small loading only; there's 500mA reserved for use by the Video Slot.

5.3.2 Clock Signals

These are various clock signals useful for synchronous timing of video peripherals.

/C1 Clock

This is a 3.58 MHz clock (3.55 MHz on PAL systems) that's synched to the falling edge of the 7M system clock. Also known as /CCK in some places.

/C3 Clock

This is a 3.58 MHz clock (3.55 MHz on PAL systems) that's synched to the rising edge of the 7M system clock. Also known as /CCKQ in some places.

/C4 Clock

This is a 3.58 MHz clock (3.55 MHz on PAL systems) that's synched to the rising edge of the 7M CDAC clock.

CDAC Clock

This is a 7.16 MHz clock (7.09 MHz on PAL systems) that trails the 7M system clock by 90°.

C28O Clock

This is a 28.64MHz clock (28.36MHz on PAL systems) that's synchonized to the digital RBG bus.

Timer Time Base (TBASE)

This is the real time clock time-base input, either 50Hz or 60Hz depending on the country involved and the setting of the Time Base Jumper. The jumper can select either line frequency or vertical synchronization as the clock's time base.

External Clock (XCLK, /XCLKEN)

The video slot provides for an external system clock, generally used to cause the entire A2000 system to become synchronized to something external. This should be something very close to the 28.64 MHz clock normally used to drive the system; the value used for XCLK can be a somewhat higher frequency, though anything too high will cause memory and other system timings to break down. XCLK will only be engaged as the system clock when /XCLKEN is asserted. There is no fixed phase relationship between XCLK and the internal clocks and video output; C28O can always be used for pixel clocking.

5.3.3 Video Signals

The main point of this slot is access to the video signals generated by the Amiga's custom video chips, may of which are not available on the 23 pin external video connector.

Analog Video (Analog Red, Analog Green, Analog Blue)

This is the analog RGB output, which consists of Red, Green, and Blue signals each of which generates a 0.7V p-p, 75 Ohm ternimated, analog output.

Digital Video (RGB$_{23}$..RGB$_0$)

These are the digital video outputs. RGB$_{23}$..RBG$_{16}$ corresponds to Red$_7$..Red$_0$, RGB$_{15}$..RGB$_8$ corresponds to Green$_7$..Green$_0$, and RGB$_7$..RGB$_0$ corresponds to Blue$_7$..Blue$_0$. These are arranged such that the high order four bits of each Red, Green, and Blue correspond to the 12 bits of digital video available in A2000 and A3000 video slots.

Separate Sync (/HSYNC, /VSYNC)

These are the separate, bidirectional, 47 Ohm terminated video frame synchronization clocks, /HSYNC is the horizontal sync, /VSYNC is the vertical sync. These sync signals are typically active low, but can be programmed active high by Alice. Genlock devices source these lines to slave the Amiga video to their input video source.

Composite Sync (/CSYNC)

/CSYNC is an unterminated, buffered, digital level composite sync. The analog composite sync line, COMPSYNC, is not available on the A3000+ video slot.

/BURST

NTSC/PAL colorburst gate, used to remove colorburst during the vertical blanking interval.

Pixel Switch (/PixelSW)

Genlock overlay color indicator, set by a variety of conditions, on a pixel by pixel basis.

SOG

This was originally a programmable sync-on-green line that went to the Kelly A/D converter. Since the demise of Kelly, sync-on-green is done via jumper, and this line is free for use by a video slot device. This line reflects the state of bit seven in BPLCON2.

Light Pen (/LPEN)

This is an input to the Alice light pen input. This signal should go low in response to the lighting of a pixel on a video display monitor. The Alice chip latches the raster position that was in effect when the /LPEN signal goes low, so an application can follow the position of a light pen on the screen.

5.3.4 Audio Signals

Along with access to video signals, audio signals are available at the Video Slot.

Audio Line Out (LineoutLF, LineoutRT)

The LineoutLF and LineoutRT lines are actually duplicated on the A3000+ Video Slot. The original line out function remains intact. However, given the addition of the DSP audio in the A3000+, a raw tap from Paula makes much less sense, so it has been eliminated.

Filter Cutoff (/LED)

This is the /LED port line which, by Amiga convention, is used to cut out the two pole low pass filter on the Paula generated audio channels. When asserted, the filter is in place; when negated the filter is bypassed. This is an input to this Video connector, useful to allow any Audio/Video card to monitor the audio filtering state.

5.3.5 Port Connections

Most of the signals from the bidirectional parallel port (printer port) are available on this connector as well, along with a few others. Note that use of the parallel port lines by a video card may conflict with the external use of this port by a printer or other device.

8 Bit Parallel Port (PD0-PD7)

The 8 bit bidirectional parallel port most commonly used to drive a Centronics interface printer externally is accessable here. It can be used to control various aspects of a complex video interface device.

Parallel Port Handshake (/PACK)

This is the acknowledge input, /PACK, the same as the acknowledge input to the parallel port. Driving this with an output from a Video Card can cause a level 2 interrupt to occur through the 8520 CIA device this is connected to, based on the programming of an 8520 register.

Other Port Lines (PBUSY, PPOUT, PSEL,/PSTROBE)

PBUSY and PPOUT are general purpose I/O signals that together can also function as a synchronous serial data port driven by an 8520 CIA device. In normal printer use, the PBUSY signal is used to indicate printer buffer full to the Amiga, while PPOUT is used to indicate the printer paper is out. For serial port usage, PBUSY is the serial clock, PPOUT is the serial data line. These should be driven with open collector devices if the Video Card uses them as inputs to the 8520. The PSEL signal is a general purpose I/O port line, usually used as a device select signal on the parallel port. /PSTROBE is a port handshake output, which can be used as a data ready or data accepted strobe by the receiving end of a parallel port transfer.

SSPB Port (SP, CNT)

The A3000+ Synchonous Serial Peripheral Bus is available here, to be used for controlling a wide variety of Audio and Video peripherals compatible with this serial bus. Since the SSPB supports multiple devices, it's a much better port to use than the parallel port for video card control as long as it's fast enough.

5.4 CD/AUX Connector

The CD/AUX connector is an eight pin SIL header, designed to allow a CD-ROM or similar device to be mixed into the A3000+ audio channels. This was originallly conceived for tower machines, which can actually house an internal CD-ROM drive, but it was left in because of its usefulness in the A3000+, even if a CD-ROM drive won't fit internally. The pinout is given in Appendix A.2.2. Basically, it provides +5V, +12V, and ground, enough to power a small chip if necessary. The Audio ground is provided for shielding, and there are left and right channel outputs, which are mixed in with the audio output from the Paula chip. Finally, the SSPB clock and data lines are available here, should a small serial bus device be useful in some kind of interface here.

5.5 DSP Connector

The A3000+ provides a 36 pin header for DSP expansion. The pinout is given in Appendix A.2.3, and additional information on the DSP signals can be found in the DSP chapter and in the DSP3210 specification, WE® DSP3210 Digital Signal Processor, Information Manual, Revision 1.8.1, from AT&T Microelectronics.

5.5.1 Power Connections

The DSP Connector provides several different voltages designed to supply DSP enhancement devices.

Digital Ground

Digital supply ground used by all digital devices in the A3000+. The DSP connector has grounds on every odd pin except the key, to facilitate the use of a short ribbon cable to the connector.

Main Supply (+5V)

Main digital level power supply, which can supply reasonably large currents. DSP enhancement devices must be factored into a full system power budget.

Negative Supply (-5V)

Negative version of the main supply, for small current loads only; there's a total of 0.3 Amp for the entire A3000+ system.

High Voltage Supply (+12V)

Higher voltage supply, intended for small loading only; this must as well be properly budgeted.

Negative High Voltage Supply (-12V)

Negative version of the high voltage supply, for small current loads only; there's a total of 0.3 Amp for the entire A3000+ system.

5.5.2 DSP Serial Bus

The raw DSP3210 serial bus is brought out to the DSP connector. A variety of serial protocols can be implemented with these lines and a PAL or so.

Frame Synchronization (SY)

This line, which can be generated by the DSP3210 or an external source, is used to synchronize frames, for serial formats that require multiple word packet frames. Internally generated frame syncs can come from either ICK or OCK.

Serial Input Port (DI, ICK, /ILD)

The DI line is the serial input. ICK is a clock driving this port, and it can sourced by the DSP3210 or an external device. /ILD is the load strobe, which can signal the start or finish of a word transfer, and can be sourced internally or externally.

Serial Output Port (DO, OCK, /OLD)

The DO line is the serial output. OCK is a clock driving this port, and it can be sourced by the DSP3210 or an external device. /OLD is the load strobe, which can signal the start or finish of a word transfer, and can be sourced internally or externally.

5.5.3 Other Signals

There is a mix of other useful signals on the DSP expansion connector.

DSP Reset (/RST)

This is the reset line which starts up the DSP.

CODEC Select (/EXP1, /EXP0)

The A3000+ logic allows one of four devices to run on the serial bus at any given time. Two are on-board, the phone line CODEC and the audio CODEC. The /EXP1 and /EXP0 lines are selects for two more, which could exist on some DSP expansion device. These lines are controlled by DSP port lines, as explained in the DSP3210 chapter.

Data/Control (D/C)

This line goes high for data mode transfers, low for control mode transfers. Not every serial bus device will have distinct data and control modes, but some do. In any case, the device running the serial bus can only change during control mode, where it is generally assumed that all devices will listen for their device selects and get off the serial bus if they are not selected.

SSPB Port (CNT, SP)

The SSPB serial port is made available here, should any low speed peripherals be useful in the design of a DSP enhancement device.

APPENDICES

A.1 External Connector Pinouts

The A3000+ has a considerable number of external ports. While not all of them are new to the A3000+, they are all covered here anyway, just for completeness.

A.1.1 Mouse Port

Pin Number	Mouse	Joystick	Other
1	V Pulse	Forward	
2	H Pulse	Back	
3	VQ Pulse	Left	
4	VH Pulse	Right	
5	Middle Button	Button 3	Analog Pot X
6	Left Button	Button 1	Light Pen Beam
7	+5V	+5V	+5V
8	Ground	Ground	Ground
9	Right Button	Button 2	Analog Pot Y

A.1.2 Keybord Port

Pin Number	Signal Name
1	No Connection
2	Data
3	Clock
4	+5V
5	Ground

A.1.3 Serial Port

Pin Number	Signal Name
1	Ground
2	TxD
3	RxD
4	RTS
5	CTS
6	DSR
7	Ground
8	CD
9	+12V
10	-12V
11	Audio Out
12-17	No Connection
18	Audio In
19	No Connection
20	DTR
21	No Connection
22	RI
23-25	No Connection

A.1.4 Parallel Port

Pin Number	Signal Name
1	/STROBE
2	Data 0
3	Data 1
4	Data 2
5	Data 3
6	Data 4
7	Data 5
8	Data 6
9	Data 7
10	/ACK
11	BUSY
12	POUT
13	SEL
14	/AUTO (+5V)
15	No Connection
16	/INIT
17-25	Ground

A.1.5 Floppy Port

Pin Number	Signal Name
1	/RDY
2	/DKRD
3-7	Ground
8	/MTRX
9	/SEL3
10	/DKRST
11	/CHNG
12	+5V
13	SIDE
14	/WPROT
15	/TRK0
16	/DKWEB
17	/DKWDB
18	STEP
19	DIR
20	No Connection
21	/SEL2
22	/INDEX
23	+12V

A.1.6 SCSI Port

Pin Number	Signal Name
1	/REQ
2	MSG
3	/ID
4	/RESET
5	/ACK
6	/BSY
7	Ground
8	/Data 0

9	Ground
10	/Data 3
11	/Data 5
12	/Data 6
13	/Data 7
14	Ground
15	/CD
16	Ground
17	/ATN
18	Ground
19	/SEL
20	/Parity
21	/Data 1
22	/Data 2
23	/Data 4
24	Ground
25	Terminal Power

A.1.7 Audio Port

Pin Number	Signal Name
1	Line In
2	Audio Ground
3	Mic In -
4	Audio Ground
5	Mic In +
6-11	Audio Ground
12	Line Out
13	+5V
14	Alternate Out
15	Audio Ground

A.1.8 Video Port

Pin Number	Signal Name
1	XCLK
2	/XCLKEN
3	Analog Red
4	Analog Green
5	Analog Blue
6	Digital Intensity
7	Digital Blue
8	Digital Green
9	Digital Red
10	/CSYNC
11	/HSYNC
12	/VSYNC
13	Ground
14	Pixel Switch
15	/C1
16-20	Video Ground
21	-5V
22	+12V
23	+5V

A.2 Internal Connector Pinouts

The A3000+ has a considerable number of external ports. While not all of them are new to the A3000+, or even generally available to the user, they are all covered here anyway for completeness.

A.2.1 Power Connector

Pin Number	Signal Name
1	Video +5V
2-5	Main +5V
6-10	Ground
11	-5V
12	+5V User
13	TICK
14	-12V
15	+12V

A.2.2 CD/AUX Connector

Pin Number	Signal Name
1	Mix in left
2	Audio Ground
3	Mix in right
4	+12V
5	+5V
6	CNT (SSPB Clock)
7	SP (SSPB Data)
8	Ground

A.2.3 DSP Connector

Pin Number	Signal Name
1,5-35 odd	Ground
2,4	+5V
6	-5V
8	+12V
10	-12V
12	SY
14	DI
16	ICK
18	/ILD
20	DO
22	OCK
24	/OLD
26	/RST
28	/EXP1
30	/EXP0
32	D/C
34	CNT (SSPB Clock)
36	SP (SSPB Data)

A.2.4 Floppy Connector

Pin Number	Signal Name
1,5-33 odd	Ground
2	/CHNG
3	KEY
4	/INUSE1
6	/INUSE0
8	INDEX
10	/SEL0
12	/SEL1
14	/INUSE1
16	/MTROD
18	DIR
20	/STEP
22	/DKWDB
24	/DKWEB
26	/TRK0
28	/WPROT
30	/DKRD
32	/SIDE
34	/RDY

A.2.5 SCSI Connector

Pin Number	Signal Name
1-17 odd	Ground
2	/Data 0
4	/Data 1
6	/Data 2
8	/Data 3
10	/Data 4
12	/Data 5
14	/Data 6
16	/Data 7
18	/Parity
19-24	Ground
25	KEY
26	Terminal Power
27-30	Ground
31-49 odd	Ground
32	/ATN
34	No Connection
36	/BSY
38	/ACK
40	/RESET
42	MSG
44	/SEL
46	/CD
48	/REQ
50	/ID

A.2.6 Daughterboard Slot

The front daughterboard slot uses the Zorro III pinout. This is the back slot, which contains signals for the extra Zorro III slots and the Video Slot on the A3000+ daughterboard.

Pin Number	Signal Name
1	Audio Line Out Left
2	Audio Line Out Right
3	C28O
4,6	+5V
5	Analog Red
7	Ground
8	+12V
9	Analog Green
10,11	Ground
12	/CSYNC
13	Analog Blue
14	/XCLKEN
15	Ground
16	/BURST
17	No Connection
18,19	Ground
20	/HSYNC
21	RGB$_{4}$
22	Ground
23	RBG$_{7}$
24	/VSYNC
25	RGB$_{15}$
26	No Connection
27	RGB$_{23}$
28	/Pixel Switch
29	-5V
30	Ground
31	XCLK
32	/C1
33	+5V
34	PSTROBE
35-37	+12V
38-40	+5V
41	Ground
42	RGB$_{20}$
43	RGB$_{21}$
44	RGB$_{22}$
45	Ground
46	RGB$_{12}$
47	RGB$_{13}$
48	RGB$_{14}$
49	Ground
50	RGB$_{5}$
51	RGB$_{6}$
52	Ground
53	SOG
54	TBASE
55	CDAC
56	PPOUT
57	/C3

58	PBUSY
59	/LPEN
60	/PACK
61	PSEL
62	Ground
63	PPD$_0$
64	PPD$_1$
65	PPD$_2$
66	PPD$_3$
67	PPD$_4$
68	PPD$_5$
69	PPD$_6$
70	PPD$_7$
71	/LED
72	Ground
73	SP (SSPB Data)
74,76	Audio Ground
75	CNT (SSPB Clock)
77	Ground
78	/EBR$_3$
79	/EBR$_2$
80	/EBR$_1$
81	/EBG$_3$
82	/EBG$_2$
83	/EBG$_1$
84	/SLAVE$_3$
85	/SLAVE$_2$
86	/SLAVE$_1$
87	RGB$_{16}$
88	RGB$_{17}$
89	RGB$_{18}$
90	RGB$_{19}$
91	RGB$_8$
92	RGB$_9$
93	RGB$_{10}$
94	RGB$_{11}$
95	RGB$_0$
96	RGB$_1$
97	RGB$_2$
98	RGB$_3$
99,100	Ground

A.2.7 Video Slot

As on the A3000, the A3000+ Video Slot is a two piece edge connector, upward compatible with A2000 and A3000 Video Slots.

A.2.7.1 Video Slot A

Pin Number	Signal Name
1	SP (SSPB Data)
2	CNT (SSPB Clock)
3	Audio Line Out Left
4	Audio Line Out Right
5	C28O
6,8	+5V
7	Analog Red
9	Ground
10	+12V
11	Analog Green
12,13	Ground
14	/CSYNC
15	Analog Blue
16	/XCLKEN
17	Ground
18	/BURST
19	/C4
20,21	Ground
22	/HSYNC
23	RGB$_4$
24	Ground
25	RBG$_7$
26	/VSYNC
27	RGB$_{15}$
28	No Connection
29	RGB$_{23}$
30	/Pixel Switch
31	-5V
32	Ground
33	XCLK
34	/C1
35	+5V
36	PSTROBE

A.2.7.2 Video Slot B

Pin Number	Signal Name
1	Ground
2	RGB$_{20}$
3	RGB$_{21}$
4	RGB$_{22}$
5	Ground
6	RGB$_{12}$
7	RGB$_{13}$
8	RGB$_{14}$
9	Ground
10	RGB$_{5}$
11	RGB$_{6}$
12	Ground
13	SOG
14	TBASE
15	CDAC
16	PPOUT
17	/C3
18	PBUSY
19	/LPEN
20	/PACK
21	PSEL
22	Ground
23	PPD$_{0}$
24	PPD$_{1}$
25	PPD$_{2}$
26	PPD$_{3}$
27	PPD$_{4}$
28	PPD$_{5}$
29	PPD$_{6}$
30	PPD$_{7}$
31	/LED
32	Ground
33	Audio Line Out Left
34,36	Audio Ground
35	Audio Line Out Right
37	RGB$_{16}$
38	RGB$_{17}$
39	RGB$_{18}$
40	RGB$_{19}$
41	RGB$_{8}$
42	RGB$_{9}$
43	RGB$_{10}$
44	RGB$_{11}$
45	RGB$_{0}$
46	RGB$_{1}$
47	RGB$_{2}$
48	RGB$_{3}$
49,50	Ground

A.2.8 Zorro III Bus Slot

Pin Number	Signal Name
1-4	Ground
5,6	+5VDC
7	/OWN
8	-5VDC
9	/SLAVE$_{N}$
10	+12VDC
11	/CFGOUT$_{N}$
12	/CFGIN$_{N}$
13	Ground
14	/C3
15	CDAC
16	/C1
17	/CINH
18	/MTCR
19	/INT$_{2}$
20	-12VDC
21	A$_{5}$
22	/INT$_{6}$
23	A$_{6}$
24	A$_{4}$
25	Ground
26	A$_{3}$
27	A$_{2}$
28	A$_{7}$
29	/LOCK
30	AD$_{8}$
31	FC$_{0}$
32	AD$_{9}$
33	FC$_{1}$
34	AD$_{10}$
35	FC$_{2}$
36	AD$_{11}$
37	Ground

38	AD$_{12}$
39	AD$_{13}$
40	Reserved
41	AD$_{14}$
42	Reserved
43	AD$_{15}$
44	Reserved
45	AD$_{16}$
46	/BERR
47	AD$_{17}$
48	/MTACK
49	Ground
50	E Clock
51	/DS$_0$
52	AD$_{18}$
53	/RESET
54	AD$_{19}$
55	/HLT
56	AD$_{20}$
57	AD$_{22}$
58	AD$_{21}$
59	AD$_{23}$
60	/BR$_N$
61	Ground
62	/BGACK
63	AD$_{31}$
64	/BG$_N$
65	AD$_{30}$
66	/DTACK
67	AD$_{29}$
68	READ
69	AD$_{28}$
70	/DS$_2$
71	AD$_{27}$
72	/DS$_3$
73	Ground
74	/CCS
75	SD$_0$
76	AD$_{26}$
77	SD$_1$
78	AD$_{25}$
79	SD$_2$
80	AD$_{24}$
81	SD$_3$
82	SD$_7$
83	SD$_4$
84	SD$_6$
85	Ground
86	SD$_5$
87-90	Ground
91	SenseZ$_3$
92	7M
93	DOE
94	/IORST
95	/BCLR
96	Reserved
97	/FCS
98	/DS$_1$
99,100	Ground

A.2.9 Local Bus/Coprocessor Slot

Pin Number	Signal Name
1	/DSACK$_1$
2,3	Ground
4	/HALT
5	R/W
6,7	Ground
8	/BGACK
9	/SBR
10,11	Ground
12	/AVEC
13	EXT90
14,15	+5V
16	/RAMSLOT
17	/BOSS
18,19	+5V
20	FC$_0$
21	/STERM
22,23	+5V
24	FC$_1$
25	/BR
26,27	+5V
28	/CBACK
29	/BERR
30	DIS_CLKS
31	/EMUL
32	/CBREQ
33	A$_8$

34	ECLK30*
35	Ground
36	A_0
37	A_9
38,39	Ground
40	A_1
41	A_{10}
42	ECLK90$_A$
43	/INT6
44	A_2
45	A_{11}
46	ECLK90
47	Ground
48	A_3
49	A_{12}
50,51	Ground
52	A_4
53	A_{13}
54	ECPUCLK$_B$
55	/WAIT
56	A_5
57	A_{14}
58	ECPUCLK$_A$
59	Ground
60	A_6
61	A_{15}
62,63	Ground
64	A_7
65	A_{16}
66	Reserved
67	/CI2P
68	A_{24}
69	A_{17}
70	Reserved
71	Ground
72	A_{25}
73	A_{18}
74,75	Ground
76	A_{26}
77	A_{19}
78	DIS_CLK30
79	/PI2C
80	A_{27}
81	A_{20}
82	Reserved
83	Ground
84	A_{28}
85	A_{21}
86,87	Ground
88	A_{29}
89	A_{22}
90	Reserved
91	/DSACK$_0$
92	A_{30}
93	A_{23}
94,95	+5V
96	A_{31}
97	/DS
98,99	+5V
100	/ECS
101	/CIOUT
102,103	+5V
104	/DBEN
105	/BG
106,107	+5V
108	/RMC
109	/CPURST
110	/FPURST
111	Reserved
112	EXTCPU
113	/EBCLR
114	/SENSEA3P
115	Ground
116	/IPEND
117	/RESET
118,119	Ground
120	/IPL$_0$
121	SIZ$_0$
122,123	Ground
124	/IPL$_1$
125	FC$_2$
126	CLK90
127	Reserved
128	/IPL$_2$
129	SIZ$_1$
130,131	Ground
132	/CIIN
133	/AS
134	/FPUCS
135	CPUCLK$_A$

136	/OCS
137	D_{31}
138,139	Ground
140	D_{15}
141	D_{30}
142,143	Ground
144	D_{14}
145	D_{29}
146	Reserved
147	/CBR
148	D_{13}
149	D_{28}
150	Reserved
151	Ground
152	D_{12}
153	D_{27}
154,155	Ground
156	D_{11}
157	D_{26}
158	Reserved
159	/BG30
160	D_{10}
161	D_{25}
162	Reserved
163	Ground
164	D_{9}
165	D_{24}
166,167	Ground
168	D_{8}
169	D_{16}
170,171	Reserved
172	D_{0}
173	D_{17}
174,175	+5V
176	D_{1}
177	D_{18}
178,179	+5V
180	D_{2}
181	D_{19}
182,183	+5V
184	D_{3}
185	D_{20}
186,187	+5V
188	D_{4}
189	D_{21}
190,191	Ground
192	D_{5}
193	D_{22}
194,195	Ground
196	D_{6}
197	D_{23}
198,199	Ground
200	D_{7}

A.3 System Configuration Jumpers

There are various strip-post jumpers (links) on the A3000 family motherboards, which control clock speed, sourcing, and other local slot related features.

J100 CLK90 Delay Jumper
This jumper has two positions. In position 1-2, it sets up CLK90 for 25MHz operation. In position 2-3, the EXT90 line drives CLK90, rather than the on-board clock logic.

J102 Board Clock
This jumper has two positions. In position 1-2, the source for CPUCLKA and CPUCLKB must be EXTCPU. In position 2-3, the 68030 and local bus clocks all derive from the same source, which is either EXTCPU or the on-board clock, depending on J104.

J103 FPU Chip Select Jumper
This jumper has two positions. With the shunt on pins 1-2, it enables the on-board FPU. With the shunt in position 2-3, disables the on-board FPU.

J104 CPU Clock
This jumper has two positions. In position 1-2, the source for CLK30 is derived from the on-board clock generator. In position 2-3, the source for CLK30 must come from EXTCLK by way of J102.

J105 System Clock Disable
In the 2-3 position, this jumper allows the DIS_CLKS line from the Local Bus Slot to operate normally. In the 1-2 position, it forces DIS_CLKS high (asserted), disabling on-board system clock generation.

J107 68030 Clock Disable
In the 2-3 position, this jumper allows the DIS_CLK30 line from the Local Bus Slot to operate normally. In the 1-2 position, it forces DIS_CLK30 high (asserted), disabling on-board 68030 clock generation.

J120 DSP Burst Select
In the 1-2 position, the DSP will not emulate 68030 burst cycles during block moves. In the 2-3 position, the DSP interface logic will run 2-cycle 68030 bursts in response to quad word block move cycles by the DSP.

J180 Lisa Configuration Options
This is a block of six jumpers that are read by Lisa via the second half of the mouse port register. None of these six jumpers currently have function assignments.

J191 Clock Type
Selects the clock type: position 1-2 for the Phillips, position 2-3 for the SGS-Thomson.

J200 NTSC/PAL Select
Selects NTSC or PAL power up for Alice.

J201 ChipRAM Size
Position 1-2 selects 2M decoding for Chip RAM, position 2-3 selects 8M decoding. The current Alice only supports 2M decoding.

J202 ROM Speed
This selects proper ROM speed. Position 1-2 sets timing for 280ns ROMs, position 2-3 sets timing for 160ns ROMs.

J350 Time Base
This selects the time base souce for the run-time clock in CIA1. Position 1-2 uses the 50/60Hz power supply tick, position 2-3 uses the vertical sync line.

J351 DF1: Recognition Code
Position 1-2 enables the automatic generation of a floppy ID code for 3.5" 880K floppy at unit 1. Position 2-3 disables ID generation by the motherboard for the unit 1 floppy drive.

J450 Sync-On-Green
Position 1-2 sets the system for standard separate sync monitors, position 2-3 sets the system for monitors that want composite sync on the analog green line.

J680 Zorro II Cache
Position 1-2 enables caching for Zorro II memory space ($00200000-$009fffff), position 2-3 disables caching for Zorro II memory space.

J852 Fast RAM Size
In the 1-2 position, RAMSEY generates control for 4MB DRAM, in the 2-3 position RAMSEY generates control for 1MB DRAM.

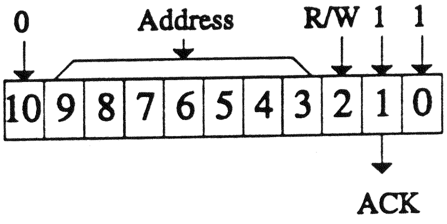
A.4 I^{2}C Bus Information

The I^{2}C Bus is a two-wire serial bus invented by Phillips, for simple connection between multiple low-speed integrated circuits. The A3000+ provides a flexible synchronous serial bus that can, when correctly configured, run I^{2}C bus transactions. The A3000+'s SSPB bus is configured for I^{2}C protocols for communication to the on-board time of day clock and audio processor. Additionally, it may be used in I^{2}C or other SSPB configurations via both DSP connector, auxiliary audio connector, and Enhanced Video Slot. For detailed information on the I2C bus, refer to "The I^{2}C Bus Specification", from Phillips Components.

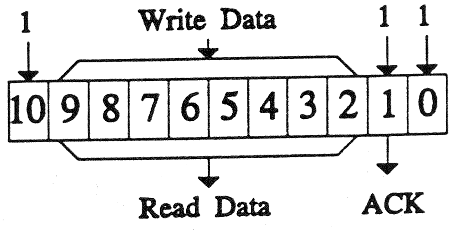
A.4.1 Configuring SSPB for I^{2}C protocols

The main protocol driver for I^{2}C on SSPB must be software, as for most SSPB protocols. Since I^{2}C defines a clock cycle not shorter than 9.4µS, for symmetric clocks, the SSPB clock register must use a value of $3B or greater to generate correct I2C clock timing. Since this will be a clocked transaction, ACNT* and ASP* should be written low, and INTENB should be high. Thus, the SSPB control register is set to $0000043B.

The I^{2}C bus has several basic data formats. The first thing a bus master (eg, the host processor, by way of the DMAC's SSPB ports) does is send out an address. I^{2}C data cycles are always eight bits long; the address cycle starts with a 7 bit address, then a bit indicating whether the next bytes will be reads or writes. Ahead of the address, though, the master needs to present a START condition, which is a high-to-low transition on the data line before any clocks take place. To pack an I^{2}C start, address, and R/W into an SSPB data word, a format is created as illustrated here. The leading bit, bit ten, is written low, to create an I^{2}C start condition. The address immediately follows, then the R/W bit. The last two bits are written high. When the SSPB word has been sent, read back the data register and examine bit one, which should have been driven low, an acknowledge, if an I^{2}C device responded to the address sent. If no device is present at that address, the acknowledge bit will return high. This is important, because it allow software to determine if a particular bus address has a device on it or not.

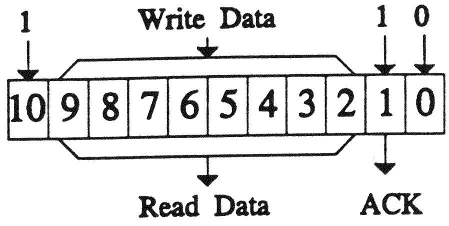


Should a particular device be found, additional I^{2}C bus cycles will send or receive data. The I^{2}C protocol only defines that the data is any number of bytes, one sent or received per bus cycle. The bus master's software should know what kind of device is at any given address, and what that device expects to send or receive in any given transaction. In any case, as mentioned, the I^{2}C protocol defines a stream of bytes that are transferred based on the bus clock. This can be encoded into the SSPB data format as shown here. Bit ten is kept high this time, with bits nine through two set to the data byte, for a write, or $FF, for a read. Bits one and zero are set high. When the



SSPB word has been sent, bit one of the data register will again contain an acknowledge bit, set low, if the transfer was successful. If this is a read, bits nine through two will also contain the read data byte.

Any number of bytes can be sent in this fashion, based only on the requirements of the device being accessed. Eventually, though, the transfer is going to be complete. The I^{2}C bus defines another special state, the STOP condition, to terminate this data transfer. In the SSPB data word, a normal I^{2}C data transfer is set up, only in this case, bit zero is set low instead of high. This will cause SP to transition from low to high after CNT is high, ending the transmission. Note also that a STOP isn't necessary if an immediate START will be following it; the START condition forces an implicit STOP. As usual, the acknowledge bit and read data, if any, can be read from the SSPB data word after transmission. Note that it is typically unnecessary to require any changes to the SSPB control word as long as one protocol type is being used on SSPB, and that protocol type fits within the constraints of the clocked transmission model, as does the I^{2}C bus.



A.4.2 A3000+ Audio Processor

The A3000+ uses an I2C bus device, the Phillips TDA8420/TDA8421 Hi-Fi Stereo Audio Processor, to help manage some of the system audio resources. This device has two stereo input channels and two stereo output channels. It has volume and stereo/mono control over both output channels, as well as bass, treble, and special effects on channel one. In the A3000+, audio directly from the Paula chip, the traditional Amiga audio, goes to channel one. On channel two is an even mix of audio from Paula and from the Audio CODEC. The output channel one goes to the headphone output on the A3000+, while output channel two is the line level output available on the audio outputs and the Enhanced Video Slot.

The device is located at bus address $40, and has eight internal eight-bit wide registers. The second byte sent to this device after the I^{2}C bus address is a secondary address byte indicating which register is being accessed. Following that byte, the actual data for the register is given. This table indicates the registers and the bit fields within them:

Addr	Name	Function	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	VL1	CH1 Volume Left	1	1	V05	V04	V03	V02	V01	V00
1	VR1	CH1 Volume Right	1	1	V15	V14	V13	V12	V11	V10
2	BA	CH1 Bass	1	1	1	1	BA3	BA2	BA1	BA0
3	TR	CH1 Treble	1	1	1	1	TR3	TR2	TR1	TR0
8	S1	Switch	1	1	MU	EFL	STL	ML1	ML0	IS
4	VL2	CH2 Volume Left	1	1	V25	V24	V23	V22	V21	V20
5	VR2	CH2 Volume Right	1	1	V35	V34	V33	V32	V31	V30
C	S2	Switch	1	1	1	1	EXS	MH1	MH0	1

The Vx5-Vx0 bits control volume for their respective channels. Volume on channel one varies from -90dB, at settings of $17 or less, to 16db at $FF, in steps of 2dB. Volume on channel two varies from -90dB at settings of $1F or less, to 0dB at $FF, in steps of 2dB. Bass control, for channel one only, varies from -12dB at $2 or less to 15dB at $B or more. Treble, also for channel one only, varies from -12dB at $2 or less to 12dB at $A or more. Both bass and treble adjustments are in steps of 3dB.

There are two switch fields. The active high Mx0 bit selects the left input for transmission to the output channel, while the active high Mx1 select the righ input for transmission to the output channel. When both are selected, stereo output will result from stereo input. The first switch field also contains the MU bit, which mutes all output when set to one (also the powerup default), passes output normally when set to zero. Also in that byte are the STL and EFL bits, which set up the special effects for channel one. Normal stereo is selected with STL high, EFL low. The "spatial" effect is selected with STL and EFL high. A pseudo-stereo effect is generated from a monoural input with the STL bit low and EFL bit high. The final bit in the first switch field is the IN bit, which selects the input source for both outputs. If IN is low, input channel one is selected, if high, input channel two is selected. Finally, switch byte two contains the EXS bit. When EXS is brought low, the external switch pin on the device is brought high impedence, when high, the external switch pin is brought to ground. The A3000+ does not use this external switch pin, but future systems may.

That is the basic description of the sound processor device. More information on this is available in the TDA8420/TDA8421 specifications from Phillips Components.

A.4.3 A3000+ RAM/Clock

The A3000+, like the A3000, supports a battery backed clock with a small amount of RAM in it. Unlike the A3000, the A3000+ uses an I^2C based RAM Clock. At the time of this writing, the final selection of the clock has not been made. The A3000+ motherboard supports both the Phillips PCF8583 clock and the SGS-Thomson MK41T56, selected via jumper J191. While the SGS part appears to be the more desirable of the two, the Phillips part appears to be the more available of the two. The clocks will appear at different I^2C bus addresses, so software can easily determine which clock is in any given A3000+ system.

A.4.3.1 The Phillips PCF8583

This device contains a low power 256x8 bit CMOS RAM and a clock with calendar function. The first 8 bytes of RAM are used for clock functions, the next 8 bytes can be used as RAM or with the alarm clock function. The clock sits at bus address $51. To access a register, the I^2C write cycle is run, with the I^2C address followed by a clock register number. The value to be written to that register follows. Successive bytes will written to successive locations in the clock RAM, thanks to an automatic address increment feature. The same can be done for reads. An I^2C write cycle is done initially, with slave address, write condition, clock RAM address, and then a new start condition. That start is followed by an I^2C read cycle. Every byte after the I^2C address will be read from the clock, using the aforementioned auto-address increment feature.

The register map is given by the following table. Note that the alarm functions really aren't used for anything in the A3000+ at present.

Addr	Function	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	Control/Status	STOP	HOLD	----FUNC----		MASK	ALEN	ALF	TIF
1	1/100 Seconds	-------------1/10th-----------				-----------1/100th----------			
2	Seconds	---------------10s-------------				---------------1s-------------			
3	Minutes	---------------10m------------				---------------1m------------			
4	Hours	12/24	PM/AM	-----10h-----		---------------1h------------			
5	Year/Date	-----YEAR-----		-----10d-----		---------------1d------------			
6	Weekday/Month	------WEEKDAY------			10M	---------------1M-----------			
7	Timer	---------------10d-------------				---------------1d------------			
8	Alarm Control	ALINT	TAEN	----ALFN-----		TIEN	---------TIFN--------		
9	Alarm 1/100 Seconds	-------------1/10th-----------				-----------1/100th----------			
A	Alarm Seconds	---------------10s-------------				---------------1s-------------			
B	Alarm Minutes	---------------10m------------				---------------1m------------			
C	Alarm Hours	12/24	PM/AM	-----10h-----		---------------1h------------			
D	Alarm Year/Date	-----YEAR-----		-----10d-----		---------------1d------------			
E	Alarm Weekday/Month	------WEEKDAY------			10M	---------------1M-----------			
F	Alarm Timer	---------------10d-------------				---------------1d------------			
10-FF	Free RAM								

The basic control register is at location zero. The STOP bit is set low for counting, high for reset. The HOLD bit is set low for founting, high to freeze the count. The FUNC field determine which function the part performs; the A3000+ sets this to 00, for standard 32.768kHz based clock. The MASK value, when high, lets the date and month count be read with year and weekday bits cleared. When MASK is low, year and weekday bits are also read. The ALEN bit is set low to disable the alarm control register, high to enable it. The ALF bit is the alarm flag, the TIF bit is the timer flag.

Most time encodings are straight binary-coded decimal. When the 12/24 bit is high, AM/PM counting is used, when low, 24 hour counting is used. The PM/AM flag goes high for PM, low for AM. The 10h field counts hours from 0 to 2. The year field counts years from 0 to 3, and the 10d field counts days 0 to 3.

Any more information on this part can be found in the Phillips data sheet, "PCF8583: Clock Calendar with 256x8-Bit Static RAM", available from Phillips Components.

A.4.3.2 The SGS-Thomson MK41T56

his device contains a low power 64x8 bit CMOS RAM and a clock with calendar function. The first 8 bytes of RAM are used for clock functions.. The clock sits at bus address $??. To access a register, the I^{2}C write cycle is run, with the I^{2}C address followed by a clock register number. The value to be written to that register follows. Successive bytes will written to successive locations in the clock RAM, thanks to an automatic address increment feature. The same can be done for reads. An I^2C write cycle is done initially, with slave address, write condition, clock RAM address, and then a new start condition. That start is followed by an I^2C read cycle. Every byte after the I^2C address will be read from the clock, using the aforementioned auto-address increment feature. The register map is given by the following table.

Addr	Function	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	Seconds	STOP	----------10s----------			---------------1s-------------			
1	Minutes	X	----------10m---------			---------------1m------------			
2	Hours	X	X	-----10h-----		---------------1h-------------			
3	Weekday	X	X	X	X	X	-----WEEKDAY----		
4	Date	X	X	-----10d-----		---------------1d-------------			
5	Month	X	X	X	10M	---------------1M------------			
6	Year	---------------10y-------------				---------------1y-------------			
7	Control	OUT	FT	S	-----------CALIBRATION------------				
8-63	Free RAM								

All the time fields are binary coded decimal, and pretty self-explanatory. When the STOP bit set brough high, it causes the clock to stop counting. Bringing it low again resumes the count. The OUT bit is the output of the final counter stage, which runs at 512Hz. When FT bit is high, the OUT signal is driven on the FT/OUT pin of the device, providing a measurable calibration point. When FT is low, the FT/OUT pin instead reflects the value of bit seven of register three.

The S bit is the sign for the CALIBRATION field. The CALIBRATION field allows the clock to be adjusted in software, rather than via a hardware trimming capacitor as in most systems. When S is one, the CALIBRATION field value increases the oscillator frequency; when S is zero, it decreases the oscillator frequency. Calibration takes place over a 64 minute cycle. The first 62 minutes in the cycle may, once per minute, have one second shortened by 128 or lengthened by 256 oscillator cycles. The value of CALIBRATION determines how many minutes get an adjusted second. Each calibration step has the effect of adding 512 or subtracting 256 oscillator cycles for every 125,829,120 actual oscillator cycles.

The clock calibration can be achieved via program control by trial and error, or by intelligent comparison against the CIA's software clock, driven by a regulated 50/60Hz. If more accuracy is desired, the aforementioned FT pin can be measured with a frequency counter.

Additional information on this part may be found in the spec sheet, "MK41T56N TimeKeeper™ RAM", from SGS-Thomson Microelectronics.

A.6 Additional CODEC Information

Some additional details on the CODEC chips are provided here. This should be sufficient information for basic programming of the devices, though in most cases, this won't be necessary, as the hardware details of each device is expected to be abstracted by a VCOS device driver. More detailed information on these devices will be provided as data sheets become available.

A.6.1 The Phone-Line CODEC

The format of the Phone-Line CODEC control registers are given in detail here. Many of the clock generation features are not currently of any importance to the A3000+.

Control Register 0

Bit(s)	Name	Explanation	Value	Effect
9	INTEN	Interpolation filter	0	Disabled
			1	Enabled
8-5	TS	TSYNC Rate	0	9600Hz
			1	8000Hz
			2	7200Hz
			3	4800Hz
			4	2400Hz
			5	1200Hz
			6	600Hz
			7	19,200Hz
			8	14,400Hz
			10	12,000Hz
4-3	SR	Sampling Rater	0	9.6kHz
			1	8.0kHz
			2	7.2kHz
2-0	OP	Clock Operating Modes	0	Asynchronous fallback
			4	V.32 TSYNC
			5	V.32 Internal Sync
			6	V.32 Loopback
			7	Async. fallback TSYNC

Control Register 1

Bits(s)	Name	Explanation	Value	Effect
9	SA87	Scale sampling 8/7	0	Disabled
			1	Enabled
4	PDA*	Analog Power-Down	0	Power-down
			1	Normal operation
3	PDD*	Digital Power-Down	0	Power-down
			1	Normal operation

2-0	FB	Filter bypass	0	No bypass
			2	ADC hi-pass filter
			3	ADC hi/lo-pass filter

Control Register 2: Receive Control

Bits(s)	Name	Explanation	Value	Effect
6-4	BA	Receive baud clock	0	2400
			1	1600
			2	1200
			3	600
3-0	BI	Receive bit-rate	0	9600
			1	8000
			2	7200
			3	4800
			4	2400
			5	1200
			6	600
			7	19,200
			8	14,400
			9	12,000
			10	19,200 unscaled by SA87

Control Register 3: Transmit Control

Bits(s)	Name	Explanation	Value	Effect
6-4	BA	Transmit baud clock	0	2400
			1	1600
			2	1200
			3	600
3-0	BI	Transmit bit-rate	0	9600
			1	8000
			2	7200
			3	4800
			4	2400
			5	1200
			6	600
			7	19,200
			8	14,400
			9	12,000
			10	19,200 unscaled by SA87

Control Register 4: ADC Phase Adjust

Bits(s)	Name	Explanation	Value	Effect
8	PHD	Phase Control	0	Phase advance
			1	Phase retard
7-0	PA	ADC Phase Adjust		

Control Register 5: DAC Phase Adjust

Bits(s)	Name	Explanation	Value	Effect
8	PHD	Phase Control	0	Phase advance
			1	Phase retard
7-0	PA	DAC Phase Adjust		

A.6.2 The Hi-Fi Audio CODEC

The format of the Hi-Fi Audio CODEC frames are given in detail here. The A3000+ expects the DSP3210 to supply the serial clocks during control frames, the CODEC to supply the serial clocks during data frames. All unused bits should be read or get written zero.

A.6.2.1 The Control Frame

The control frame is logically broken up into eight byte-wide logical registers. The bytes are arranged from slot 0 to slot 7, MSB first, to create the control frame.

Control Slot 0: Status Register

Bits(s)	Name	Explanation	Value	Effect
2	DCB	Data control handshake		
0	AC	Force autocalibration	1	Write 1 to force

Control Slot 1: Data Format Register

Bits(s)	Name	Explanation	Value	Effect	
5-3	DFR	Data conversion frequency (depends on MCK)	0	8.00000kHz	5.51250kHz
			1	16.00000kHz	11.02500kHz
			2	27.42857kHz	18.90000kHz
			3	32.00000kHz	22.05000kHz
			4	N/A	37.80000kHz
			5	N/A	44.10000kHz
			6	48.00000kHz	33.07500kHz
			7	9.60000kHz	6.61500kHz
2	ST	Mono/stereo	0	Mono	
			1	Stereo	
1-0	DF	Data format	0	16-bit linear	
			1	8-bit µ-law	
			2	8-bit A-law	
19	MCK	Master clock	0	Serial clock	
			1	XTAL1, 24.5760MHz	
			2	XTAL2, 16.9344MHz	

Control Slot 2: Serial Port Control Register

Bits(s)	Name	Explanation	Value	Effect
5-4	MCK	Master clock source	0	Serial clock
			1	XTAL1, 24.5760MHz
			2	XTAL2, 16.9344MHz
3-2	FSEL	Frame size select	0	64 bits/frame
			1	128 bits/frame
			2	256 bits/frame
1	XCLK	Transmit clock	0	Use external clocks
			1	Generate clocks
0	XEN	Transmitter enable	0	Enable serial output
			1	Disable serial output

Control Slot 3: Test Register

Bits(s)	Name	Explanation	Value	Effect
1	ADL	Loopback mode	0	Digital loopback mode
			1	Analog loopback mode
0	ENL	Loopback enable	0	Loopback disabled
			1	Loopback enabled

Control Slot 4: Parallel Port Register

Bits(s)	Name	Explanation	Value	Effect
7	PIO1	Parallel I/O 1		
6	PIO0	Parallel I/O 0		

Control Slot 6: Version Register

Bits(s)	Name	Explanation	Value	Effect
3-0	RV	Device revision code	0	

A.6.2.2 The Data Frame

The data frame is logically broken up into eight byte-wide logical registers. The bytes are arranged from slot 0 to slot 7, MSB first, to create the data frame. For eight-bit frames, only the MSBs are used. For mono frames, only the Left channel is used.

Data Slot 0-1: Left Audio

Bits(s)	Name	Explanation	Value	Effect
15-0	LEFT	Left Channel Audio (MSB/LSB)		

Data Slot 2-3: Right Audio

Bits(s)	Name	Explanation	Value	Effect
15-0	RIGHT	Right Channel Audio		

Data Slot 4: Ouput Control

Bits(s)	Name	Explanation	Value	Effect
7	LE	Line output enable	0	Analog line output off
			1	Analog line output on
6	HE	Headphone output enable	0	Headphone output off
			1	Headphone output on
5-0	LO	Left channel attenuation	0	No attenuation
			62	93dB attenuation
			63	Digital mute

Data Slot 5: Ouput Control

Bits(s)	Name	Explanation	Value	Effect
6	SE	Speaker output enable	0	Speaker off
			1	Speaker on
5-0	RO	Right channel attenuation	0	No attenuation
			62	93dB attenuation
			63	Digital Mute

Data Slot 6: Input Control

Bits(s)	Name	Explanation	Value	Effect
7	PIO1	Parallel I/O 1		
6	PIO0	Parallel I/O 0		
5	OVR	Audio level overrange	0	Within range
			1	Over range
4	IS	Input select	0	Line level input
			1	Microphone input
3-0	LG	Left input gain	0	No gain
			15	22.5dB gain

Data Slot 7: Input Control

Bits(s)	Name	Explanation	Value	Effect
7-4	MA	Monitor path attenuation	0	No attenuation
			14	84dB attenuation
			15	Monitor path muted
3-0	RG	Right input gain	0	No gain
			15	22.5dB gain

A.6.2.3 Reset Conditions

The Audio CODEC is reset by the A3000+ DSPRST* signal. The control register parameters on powerup are given here.

Register	Value
MCK	0
AC	0
LE	0
IS	0
SRE	0
DF	1
XCLK	0
ADL	0
DCB	1
MA	15
SE	0
ST	0

Register	Value
LO	63
RO	63
RG	0
OVR	0
DFR	0
FSEL	2
ENL	0
HE	0
LG	0
PIO1	1
PIO0	1
XEN	1

A.7 References

MC68030 Enhanced 32 Bit Microprocessor User's Manual, Second Edition, Motorola Inc., number MC68030UM/AD REV1.

WE® DSP3210 Digital Signal Processor, Information Manual, Revision 1.8.1, AT&T Microelectronics.

COM20020 Universal Local Area Network Controller (ULANC), Standard Microsystems Corporation, Component Products Division.

WD33C93A SCSI Bus Interface Controller, Western Digital Corporation.

WD33C93B (SBIC) Enhanced SCSI Bus Interface Controller, Western Digital Corporation.

The I^{2}C Bus Specification, Phillips Components.

TDA8420: Hi-Fi Stereo Audio Processor; I^{2}C Bus, Phillips Components.

PCF8583: Clock Calendar with 256x8-Bit Static RAM, Phillips Components.

MK41T56N TimeKeeper™ RAM, SGS-Thomson Microelectronics.

The Zorro III Bus Specification, Revision 1.10, Commodore-Amiga, Inc.

The A3000 Local Bus Slot, Revision 2.00, Commodore-Amiga, Inc.

Overview of the AA Chip Set

by Chris Green

When the Amiga was introduced in 1985, its graphics capabilities far surpassed those of other home computers. It supported the NTSC and PAL video standards without requiring expensive add-ons. Its display co-processor, blitter, sprites, scrolling, and dual-playfield mode gave it animation capabilities which rivaled dedicated arcade machines. Its 4096 color palette and HAM mode made it the first inexpensive computer capable of displaying good quality digitized and rendered pictures.

The first upgrade to this chip set was the ''Fat Agnus'' part, which was introduced to satisfy user's needs for more chip RAM.

The next upgrade was the ECS Denise, which addressed the need for higher quality video titling resolution, enhanced genlock capability, and non-interlaced displays for productivity applications.

Commodore also created the A2024 high-resolution monochrome display for desktop publishing and other applications, and the display enhancer for 31khz full color output.

However, Amiga users still clamor for more colors and resolution, and now we have a new chip set to address their concerns. This chip set, referred to as AA (pronounced ''double A''), satisfies the desire for more colors, a bigger palette, and other enhancements, while remaining register level compatible with the old chip set, and remaining true to the spirit of the original design.

Terminology

Throughout these notes, the following terms are used:

31Khz, 15Khz - These refer to the frequency of the horizontal blanking of the display. 15khz is the normal (or interlaced) amiga frequency. 31khz is the frequency used when displaying non-interlaced 400+ line modes. 31khz is what you get out of a display enhancer or in productivity mode.

Lores/Hires/SuperHires - these terms are used to refer to the apparent pixel size on the screen. Lores is 1/320th of a scan line, Hires is 1/640th, and SuperHires is 1/1280th.

Productivity - this is a 31khz mode which display Hires-sized pixels. Pixels are clocked at 35ns.

35ns/70ns/140ns - These refer to the period of the pixel clock, in nanoseconds. In a 15khz mode, 35ns is SuperHires, 70ns is Hires, and 140ns is Lores. In a 31khz mode, 35ns is Hires, 70ns is Lores, and 140ns is ExtraLores (1/160th of the screen width pixels).

1x/2x/4x - One of the major features of the new chips is the enhancement of display memory bandwidth. By using 32-bit burst mode fetches, AA systems can quadruple the speed at which display data is fetched. A system with this capability is referred to as a 4x system. A 2x system is one that either does 16 bit burst-mode fetches or 32 bit non-burst fetches. A 1x system does 16 bit non-burst fetches (like the current amiga chips).

Extra halfbright - a mode which was used on the original amiga chip set to provide 64 colors on a lores screen. While this mode is still present, it is a good deal less useful now that there is a 256-color palette.

New Display Modes

The AA chips support many more possible display resolutions and color combinations than the previous chip sets. Also, most restrictions on allowable combinations no longer exist. The number of bitplanes available in all modes has been increased to 8. HAM and extra halfbright modes are now available in Hires and SuperHires resolutions. The following modes are supported:

For 4x systems:

SuperHires	1-8 bitplanes
Productivity	1-8 bitplanes
Hires	1-8 bitplanes
Lores	1-8 bitplanes

For 2x systems:

SuperHires	1-4 bitplanes
Productivity	1-4 bitplanes
Hires	1-8 bitplanes
Lores	1-8 bitplanes

For 1x systems:

SuperHires	1-2 bitplanes
Productivity	1-2 bitplanes
Hires	1-4 bitplanes
Lores	1-8 bitplanes

An 8 bitplane HAM mode has been added. This enhances the old 6 bit HAM mode by allowing the use of 64 base colors. The modify fields are now 6 bits, thus permitting the display of 256K colors at once. In fact, since the lower 2 bits of the RGB value are left alone when doing a ''modify'' operation, it is possible to display images with a 24-bit color resolution, given good choices for color register values. The new 8 bit plane HAM mode works in all display resolutions, provided enough memory bandwidth is there.

Resolution of bitplane scrolling and display window positions can now be specified in 35ns resolution, regardless of display mode. For instance, you could scroll a Lores screen in 1/4 pixel increments, thus providing the illusion of enhanced resolution.

A new register has been added to allow collision control over the full eight bitplanes.

In dual-playfield modes, 2 four-bitplane playfields are now possible in all resolutions.

Color Palette Enhancements

The color palette has been expanded from 32 colors out of 4096 to 256 colors out of 16 million. Each of these 256 color registers also has a bit to turn genlock transparency on or off.

New registers provide the capability to change the offset in the color table for the two playfields, and the odd and even sprites. For instance, you could have a 32 color screen with the bitplanes displaying the lower 32 colors, while the sprite colors came from colors 128 and up. These registers could also be used for color cycling.

The reduced color palette in SuperHires and Productivity modes has been removed. Productivity and SuperHires can now use the full 24-bit palette range.

A new bit allow reading of the color palette.

New Sprite Features

Sprites have also been enhanced to take advantage of the increased memory bandwidth. Sprites can be 32 bits wide in a 2x system, and 64 bits wide in a 4x system. The layout of sprite data in these modes is somewhat different. The position information is always at offset zero in the sprite data structure. 16 bit sprites have the control data at offset 2; 32 bit ones have it at offset 4, and 64 bit sprites have it at offset 8.

Sprite positioning resolution is now improved by a factor of 4. In a superHires or Productivity screen, the sprite can point at each pixel. In a Hires screen, it can point at each half-pixel, and every quarter-pixel in Lores. The new low bits of the x position are stored in bits 3 and 4 of the sprite control register.

Sprites can now be displayed with double or quadruple resolution, independently of the screen's resolution. For instance, you could have a Hires sprite on a Lores screen.

Changes in sprite width and resolution affect all sprites.

A new bit now allows sprites to be displayed in the border regions of the screen.

Attached sprites now work in SuperHires and Productivity modes.

Scan-Doubling

One shortcoming of the original ECS productivity mode is that when a non-interlaced screen is exposed behind a Productivity one, that screen's aspect ratio would be wrong. Also, sprites look odd in productivity mode, because their aspect ratio may not match the aspect ration they were designed for.

To address these problems, and to make Productivity mode a better substitute for the display enhancer, scan-doubling features have been added. These allow for the sprite's apparent vertical size to be doubled by setting a bit. Non-interlaced 200/256-line screens can now also have their visual size doubled by displaying each scan-line twice.

Compatibility

At powerup time, the register settings of the AA chips are 100% compatible. The standard caveats apply - write unused bits as zero, and ignore unused bits on reads.

Migrating to 2.0

by Adam Levin

Version Checking

This is used for accessing new features of existing libraries. Libraries have a version number that is easily accessible through the library base (returned by the OpenLibrary() call). For example:

```
#define LIBRARY_VERSION 36L

    /* for an imaginary library */
    LibBase = OpenLibrary("libname.library", LIBRARY_VERSION);
    if (NULL != LibBase)
        lib_version = LibBase->lib_Version;
```

The version, lib_Version, is part of the Library structure (see <*exec/libraries.h*>). Most libraries have a more complicated library base, but they always begin with a Library structure. For instance, in *graphics.library*, the version number is:

```
extern struct GfxBase *GfxBase;
lib_version = GfxBase->LibNode.lib_Version;
```

As always, the library base is not valid until the library has been opened. Note that opening the library with a version number of 0 will open any version of that library. If you require a specific version of a library, you can either:

Specify the minimum version number in the OpenLibrary() call. Note that the library will not open if the library's version number is less than the specified version, or

Test the library version after the library is open.

CAUTION: When you test for a specific version of a library, you usually are trying to open a library with a version number at least as high as some minimum version number. Thus, unless you only want a single, specific version of a library, you should test as follows:

```
/* test for at least the required version of the library */
if (LibBase->lib_Version >= LIBRARY_VERSION)
{
    /* place version-specific code here */
}
```

If you test only for equality, the test will no longer succeed with newer versions of Kickstart. Note that there is an assumed minimum version of 33, which is defined in <*exec/types.h*> as `LIBRARY_MINIMUM`.

Unavailable Libraries

This works best for libraries that were not available in a previous release. If you get the library open, then you know that you have access to all current features. For example, using the ASL library you might have code such as:

```
#include <libraries/asl.h>

#define ANY_VERSION 0L

struct Library *AslBase;

    /* Open the library and get the library base.
    ** Check for success before using it.
    ** "AslName" is #defined in <libraries/asl.h>.
    */
    AslBase = OpenLibrary(AslName, ANY_VERSION);

    /* somewhere later on in the code... */
    if (NULL != AslBase)
    {
       /* here the library is open...use asl library */
    }
    else
    {
       /* here the library failed to open,
       ** fall back to old code for requesters.
       */
    }

    /* before we quit, be sure to close it... */
    if (NULL != AslBase)
         CloseLibrary(AslBase);
```

Specific libraries to which this applies:

ASL	File requester and font requester.
iffparse	IFF handling routines.
gadtools	Intuition front end (gadgets and menus).

Version Strings

Many software buyers want to be reassured that they have the latest version of your software. The new AmigaDOS *Version* command will now search any specified file for a specially formulated string, and then print out the information contained in that string.

In 'C' code, a version string need only be declared like so:

```
UBYTE vers[] = "$VER: ProgName 36.181 (07.09.91)";
```

to become embedded in the resulting executable.

In an ASCII file, simply include a string like the one within the quotation marks above somewhere in the file.

A leading null character (0x00) used to be required, but has been made optional to allow use in all ASCII text files. ProgName is the name of the program (version ignores the file name), and 36.181 is the specific version number. The version number is fairly free format, and (for the moment) may contain alphabetic characters as well (for instance, "3.2a"). The date must be in the form (DD.MM.YY) and while it will not be displayed by *Version* it will be visible if "Type OPT H" is used on the file. Both the version number and date are updated by the *BumpRev* utility available on the companion DevCon disk.

Note that the version command will only print out the first version string found.

Pen Array

2.0's "new look" adds a three dimensional effect to the display. This effect depends on knowing which color or pen to use for such items as highlight (shine), shadow, background, etc. Rather than hard coding certain pens and requiring them to have a specific relationship to each other, there is now a pen array defined in intuition. This array defines which pen is used for a given feature, and allows Intuition to properly display information (titles, menus, gadgets...) under a wide variety of palettes.

You can use "hard coded" pen values for V33, but you should get the screens pen array for V36 (GetScreenDrawInfo()). Some things to watch out for:

1. Opening a screen with an empty array:

```
UWORD *pens = { ~0, };
```

does not necessarily get you the current Workbench pen array. What it does get you is one of two system default pen arrays, one designed for monochrome screens (1 bit plane) and one designed for multi-color screens. The multi-color array only assumes a four color screen, other colors are not used by the pen array.

2. To get the Workbench pen array, you have to specifically call GetScreenDrawInfo() with a pointer to the Workbench screen. This pen array is only valid on screens that have at least as many bitplanes as the Workbench screen, and the palette matches the Workbench screen. You can get a pointer to the Workbench screen by either opening a window on the Workbench screen, or by using LockPubScreen().

While there is currently no facility for changing the pens of the Workbench, this capability may be added in the future. Do not assume that the Workbench screen pens will match the system default screen pens.

3. You may wish to add a preferences editor that allows the user to adjust the pen array for the application's multi-color screen. This would be most important for screens with more than two bitplanes, so that the display will use the available colors.

The file *<intuition/screens.h>* defines a number of rendering pens which define the values in the pen array. These "defines" are used as indexes into the pen array (DrawInfo.dri_Pens[]).

`DETAILPEN`	compatible pen
`BLOCKPEN`	compatible pen
`TEXTPEN`	text on background
`SHINEPEN`	bright edge on 3D objects
`SHADOWPEN`	dark edge on 3D objects
`FILLPEN`	active window/selected gadget fill
`FILLTEXTPEN`	text over `FILLPEN`
`BACKGROUNDPEN`	always color 0
`HIGHLIGHTTEXTPEN`	special color text, on background

Also see *<intuition/screens.h>* for the definition of the DrawInfo structure (returned by GetScreenDrawInfo()).

"Read More About It" in these Commodore publications:

Migrating to 2.0 Issues:
"The ASL Library", *Amiga Mail Volume Two,* page I-7
"An Introduction to V36 Screens and Windows", *Amiga Mail Volume Two,* page IV-3
"Opening Screens and Windows on any Amiga", *Amiga Mail Volume Two,* page IV-17
"Tag Items and Tag Lists", *Amiga Mail Volume Two,* page I-1
"2.0 Version Strings", *Amiga Mail Volume Two,* page I-3

Compatibility Issues:
"68040 Compatibility Warning", *Amiga Mail Volume Two,* page III-11
"(2.0) Compatibility", *1990 Developer's Conference Notes,* Section 16

PAL/NTSC Issues:
"PAL-NTSC Switch for the A2000", *Amiga Mail Volume One,* I-31
"PAL and NTSC Differences", *Amiga Mail Volume One,* I-9
"PAL and Audio", *Amiga Mail Volume One,* VI-1
"PAL and NTSC Genlock Interface Guidelines", *Amiga Mail Volume One,* page XII-27

Also recommended is "Crossing Borders", Commodore's soon-to-be-released guide to software internationalization (available from CATS).

```
;/* GetFileName.c - Execute me to compile me with SAS/C v5.10a
LC -b1 -cfistq -v -y -j73 getfilename.c
Blink FROM LIB:c.o,getfilename.o TO getfilename LIBRARY LIB:LC.lib,LIB:Amiga.lib
quit

This simple example provides a subroutine getfilename(buffer,win,mode) which
gets a filename using the ASL file requester if asl.library is available.
Otherwise it calls oldgetfilename(buffer,win,mode) which could call your old
file request code (here it just prompts for a filename from stdin).

Mode is 0 (MODE_LOAD) for a load, and 1 (MODE_SAVE) for a save.

If a window pointer is provided, the requester will come up on that window's
screen.  This example first opens a load requester, then a save requester.
A custom title string and OK gadget string are used.
*/

#include <exec/types.h>
#include <exec/memory.h>
#include <dos/dos.h>
#include <intuition/intuition.h>
#include <libraries/asl.h>

#ifdef LATTICE
#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/asl_protos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


int CXBRK(void) { return(0); }  /* Disable Lattice CTRL/C handling */
int chkabort(void) { return(0); }  /* really */
#endif

/**********    debug macros      ***********/
#define MYDEBUG     0    /* Set to 1 for debugging output. */
void kprintf(UBYTE *fmt,...);
void dprintf(UBYTE *fmt,...);
#define DEBTIME 0
#define bug printf
#if MYDEBUG
#define D(x) (x); if(DEBTIME>0) Delay((ULONG)DEBTIME);
#else
#define D(x) ;
#endif /* MYDEBUG */
/********** end of debug macros **********/

#define MINARGS 1
#define ANY_LIB_VER 0L

UBYTE *vers = "$VER: getfilename 37.3 (07.08.91)";
UBYTE *Copyright =
"getfilename v37.3\nCopyright (c) 1990-91 Commodore-Amiga, Inc.  All Rights
Reserved";
UBYTE *usage = "Usage: getfilename";

/* protos */
UBYTE  *getfilename(UBYTE *copybuf, struct Window *win, int mode);
UBYTE  *oldgetfilename(UBYTE *copybuf, struct Window *win, int mode);
BOOL   AslRequestTags( APTR request, ULONG tags, ...);
void bye(UBYTE *s, int e);
void cleanup(void);

/* for file requester */
#define MODE_LOAD 0
#define MODE_SAVE 1
#define NBUFSZ (256)
UBYTE fnamebuf[NBUFSZ] = {0};
#define INITDIR "SYS:"
```

```
struct Library    *AslBase = NULL;
struct FileRequester *freq = NULL;

BOOL FromWb;

void main(int argc, char **argv)
{
    UBYTE          *filename;

    FromWb = (argc == 0) ? TRUE : FALSE;

    if ((argc) && (argc < MINARGS))
    {
        printf("%s\n%s\n", Copyright, usage);
        bye("", RETURN_OK);
    }


    AslBase = OpenLibrary("asl.library", ANY_LIB_VER);  /* used if available */

    /* if on a custom screen, we'd pass window as second arg */

    printf("\nHere is a load requester.\n");
    printf("When requester appears, go to any volume:dir and select a file\n");
    Delay(1L * TICKS_PER_SECOND);

    if (filename = getfilename(fnamebuf, NULL, MODE_LOAD))
    {
        printf("filename for load is %s\n", fnamebuf);
    }
    else
        printf("No filename selected for load.\n");


    printf("\nHere is a save requester (allows directory creation).\n");
    printf("\nNote that the previous directory and file are remembered.\n");
    Delay(1L * TICKS_PER_SECOND);

    if (filename = getfilename(fnamebuf, NULL, MODE_SAVE))
    {
        printf("filename for save is %s\n", fnamebuf);
    }
    else
        printf("No filename selected for save.\n");

    bye("", RETURN_OK);
}

void bye(UBYTE * s, int e)
{
    if (*s)
    {
        printf("%s\n", s);
        if (FromWb)
        {
            printf("Press <RET> to exit: ");
            getchar();
        }
    }
    cleanup();
    exit(e);
}

void cleanup()
{
    if (freq)
        FreeAslRequest(freq);
    if (AslBase)
        CloseLibrary(AslBase);
}
```

```
UBYTE           *
getfilename(UBYTE * copybuf, struct Window * win, int mode)
{
    ULONG           funcflags;
    UBYTE          *hail, *oktext;

    hail = (mode == MODE_LOAD) ? "Load File" : "Save File";
    oktext = (mode == MODE_LOAD) ? "Load" : "Save";
    funcflags = (mode == MODE_SAVE) ? FILF_SAVE : 0;
    funcflags |= ((win) && (win->UserPort)) ? 0 : FILF_NEWIDCMP;


    D(bug("In getfilename\n"));

    /* If have asl.library, try requester */
    if (AslBase)
    {
        /* if we don't have one, alloc it */
        if (!freq)
            freq = AllocAslRequest(ASL_FileRequest, NULL);

        if (freq)
        {
            D(bug("About to Request File\n"));

            if (AslRequestTags((APTR) freq,
                               ASL_Hail, hail,
                               ASL_OKText, oktext,
                               ASL_Window, win,
                               ASL_FuncFlags, funcflags,
                               TAG_DONE))
            {
                D(bug("Request File successful\n"));
                strcpy(copybuf, freq->rf_Dir);
                if (AddPart(copybuf, freq->rf_File, NBUFSZ))
                    return (copybuf);
            }
            else
                return (NULL);
        }
    }
    else
        return (oldgetfilename(copybuf, win, mode));
}


/*
 * Replace this with code that calls your old file requester or brings up a
 * string gadget, etc.
 */
UBYTE           *
oldgetfilename(UBYTE * copybuf, struct Window * win, int mode)
{
    printf("\nEnter filename, or <Return> to exit: ");
    copybuf[0] = '\0';
    gets(copybuf);
    return (copybuf[0] ? copybuf : NULL);
}

/*----------------------------------------------------------------------*/


BOOL
AslRequestTags(request, tags)
    APTR            request;
    ULONG           tags;
{
    return (AslRequest(request, (struct TagItem *) & tags));
}
```

```
/*
 * ExtScreen - An example of opening a Screen and a Window in an operating
 * system version independent manner. SAS/C compile and link with: LC -bl
 * -cfist -L -v -y ExtScreen.c
 */

#include <exec/types.h>
#include <intuition/intuition.h>
#include <intuition/screens.h>
#include <graphics/gfxbase.h>
#include <graphics/displayinfo.h>
#include <stdio.h>
#include <stdlib.h>

#include <clib/exec_protos.h>
#include <clib/intuition_protos.h>
#include <clib/graphics_protos.h>

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
extern struct ExecBase *SysBase;

UBYTE           vers[] = "$VER: ExtScreen 1.1 (07.08.91)";

/* Define the name of Intuition's library if it hasn't itself. */
#ifndef INTUITIONNAME
#define INTUITIONNAME "intuition.library"
#endif

/*
 * To get that New Look we'll use Workbench compatible pen numbers. The end of
 * the pen array is indicated with ~0. Note that the screen depth should comply
 * with the pen numbers used. I.e. specifying pen number 3 on a one bitplane
 * screen will obviously not give the desired effect. A description of the pen
 * array can be found in intuition/screens.h.
 */

static UWORD    dri_Pens[] = {0, 1, 1, 2, 1, 3, 1, 0, 3, ~0};

/* zoom/zip/zap/zop array to illustrate NW_EXTENDED */
static UWORD    zoomdata[] = {30, 30, 200, 100};

/* Old style requester to be compatible with < V36 */
struct IntuiText reqtext[] =
{
    {0, 1, JAM1, 10, 10, NULL, NULL, NULL},
    {0, 1, JAM1, 6, 3, NULL, "Continue", NULL},
};

/* local protos */
void            main(void);
BOOL            CheckPAL(STRPTR screenname);

void main(void)
{
    struct ExtNewScreen xnewscreen;
    struct ExtNewWindow xnewwindow;
    struct Screen  *screen;
    struct Window  *window;
    struct IntuiMessage *msg;
    struct DisplayInfo displayinfo;
    struct TagItem  taglist[3];
    BOOL            OpenA2024 = FALSE;
    BOOL            IsV36 = FALSE;
    BOOL            IsPAL;

    if (IntuitionBase = OpenLibrary(INTUITIONNAME, LIBRARY_MINIMUM))
    {
        if (GfxBase = (struct GfxBase *)
            OpenLibrary(GRAPHICSNAME, LIBRARY_MINIMUM))
        {
```

```
/*
 * V36 check if a 10 Hz A2024 screen can be opened. If this has to
 * be done in V35 (Jumpstart), start by checking the library
 * version ofcourse. Next simply open the screen with A2024
 * parameters. If this fails there are a couple of possibilities.
 * The system may have run out of memory or even though the user is
 * running V35 s/he hasn't set up Hedley mode. Logical step in case
 * of failure is to try again with regular parameters. Depending on
 * the nature of the program, this could be done everytime the
 * program starts, or according to some kind of user preferences
 * setting.
 */

/* Check library version. If >= 36 check Hedley availability */
if (GfxBase->LibNode.lib_Version >= 36)
{
    IsV36 = TRUE;

    /*
     * Use GetDisplayInfoData() with the DTAG_DISP to get the
     * display info, containing availability. Note that
     * availability means that the user admonitor'ed A2024, not
     * necessarily that there is a physical A2024 out there.
     */
    if (GetDisplayInfoData(NULL, (UBYTE *) & displayinfo,
        sizeof(struct DisplayInfo), DTAG_DISP, A2024TENHERTZ_KEY))
        if (displayinfo.NotAvailable == 0)
            OpenA2024 = TRUE;
}
else
{

    /*
     * Check if V35. If it is indicate I want the  tags added to
     * open a A2024 screen. Note that this still doesn't mean I can
     * actually open in Hedley mode.
     */
    if (GfxBase->LibNode.lib_Version == 35)
        OpenA2024 = TRUE;
}

/*
 * Use a separate CheckPAL() function to see if how the use has set
 * up the system.
 */
IsPAL = CheckPAL("Workbench");

/*
 * Build taglist, this is completely ignored in V33/V34 and V35
 * doesn't recognize tags other than those A2024 ones.
 */

/*
 * Pass the A2024 tags. Note that with V36 it is easy to pass the
 * displaymodeid as a tag (SA_DisplayID) that that is not V35
 * compatible. Also for V35 compatability this must be the first
 * tag.
 */

if (OpenA2024)
{
    taglist[0].ti_Tag = NSTAG_EXT_VPMODE;
    taglist[0].ti_Data = VPF_A2024 | VPF_TENHZ;
}
else
{

    /*
     * With V36 Intuition fully supports overscan. We'll pass the
     * Overscantype used by Workbench as a tag, and specify
     * STDSCREENWIDTH/HEIGHT in the Width and Height fields of the
```

```
     * ExtNewScreen structure.
     */
    taglist[0].ti_Tag = SA_Overscan;
    taglist[0].ti_Data = OSCAN_TEXT;
}

/*
 * Indicate we want the New Look if this system is running V36 by
 * specifying the SA_Pens tag and passing the pen array as data.
 */
taglist[1].ti_Tag = SA_Pens;
taglist[1].ti_Data = (ULONG) dri_Pens;

/* End the taglist with TAG_DONE */
taglist[2].ti_Tag = TAG_DONE;


/* If V36, the overscan mode will give us the right offsets */
xnewscreen.LeftEdge = 0;
xnewscreen.TopEdge = 0;

/* Width = 1008 if A2024, 640 if < V35, STDSCREENWIDTH if V36> */
xnewscreen.Width =
    (OpenA2024) ? 1008 : (IsV36) ? STDSCREENWIDTH : 640;

/*
 * Height = 1024 if A2024 & PAL, 800 if A2024 & NTSC, else
 * STDSCREENHEIGHT
 */
xnewscreen.Height =
    (OpenA2024) ? ((IsPAL) ? 1024 : 800) : STDSCREENHEIGHT;

xnewscreen.Depth = 2;
xnewscreen.DetailPen = 0;
xnewscreen.BlockPen = 1;

/* Set viewmodes to 0 if going to attempt to open in Hedley mode */
xnewscreen.ViewModes = (OpenA2024) ? 0 : HIRES | LACE;

/* Use NS_EXTENDED to tell V35 tags are on their way */
xnewscreen.Type = CUSTOMSCREEN | NS_EXTENDED;

/* Default font */
xnewscreen.Font = NULL;

xnewscreen.DefaultTitle =
    (OpenA2024) ? "VPF_A2024|VPF_TENHZ" : "HIRES|LACE";
xnewscreen.Gadgets = NULL;

/*
 * Pass the taglist as a V35 compatible extension. V34 will ignore
 * this
 */
xnewscreen.Extension = taglist;

if ((screen = OpenScreen(&xnewscreen)) == NULL)
{

    /*
     * Can't open screen. Might be V35 A2024 failure. Try with
     * something simpler.
     */
    xnewscreen.Width = (IsV36) ? STDSCREENWIDTH : 640;
    xnewscreen.Height = STDSCREENHEIGHT;
    xnewscreen.ViewModes = HIRES | LACE;
    xnewscreen.DefaultTitle = "HIRES|LACE";
    /* Get rid of A2024 tags, keep the others. */
    taglist[0].ti_Tag = SA_Overscan;
    taglist[0].ti_Data = OSCAN_TEXT;
    OpenA2024 = FALSE;
    screen = OpenScreen(&xnewscreen);
```

```
                /* If it still fails, give up */
            }

            /* If screen opened, open a simple ExtNewWindow on it and wait */
            if (screen)
            {
                /* Give me a zoom gadget on my window. */
                taglist[0].ti_Tag = WA_Zoom,
                    taglist[0].ti_Data = (ULONG) zoomdata;
                taglist[1].ti_Tag = TAG_DONE;

                xnewwindow.LeftEdge = 0;
                xnewwindow.TopEdge = screen->BarHeight + 1;
                xnewwindow.Width = screen->Width;
                xnewwindow.Height = screen->Height - xnewwindow.TopEdge;
                xnewwindow.DetailPen = 0;
                xnewwindow.BlockPen = 1;
                xnewwindow.IDCMPFlags = CLOSEWINDOW;
                xnewwindow.Flags =
                    WINDOWSIZING | WINDOWDRAG | WINDOWDEPTH | WINDOWCLOSE |
                    NW_EXTENDED | SMART_REFRESH | NOCAREREFRESH | ACTIVATE;
                xnewwindow.FirstGadget = NULL;
                xnewwindow.CheckMark = NULL;
                xnewwindow.Title = "Close to exit.";
                xnewwindow.Screen = screen;
                xnewwindow.BitMap = NULL;
                xnewwindow.MinWidth = 100;
                xnewwindow.MinHeight = 50;
                xnewwindow.MaxWidth = ~0;
                xnewwindow.MaxHeight = ~0;
                xnewwindow.Type = CUSTOMSCREEN;
                /* The window extension is completely ignored if not V36 */
                xnewwindow.Extension = taglist;

                if (window = OpenWindow(&xnewwindow))
                {
                    WaitPort(window->UserPort);
                    while (msg = (struct IntuiMessage *)
                                GetMsg(window->UserPort))
                        ReplyMsg((struct Message *) msg);
                    CloseWindow(window);
                }
                else
                {
                    reqtext[0].IText = "Can't open window";
                    AutoRequest(NULL, &reqtext[0], NULL,
                                &reqtext[1], NULL, GADGETUP, 320, 60);
                }
                CloseScreen(screen);
            }
            else
            {
                reqtext[0].IText = "Can't open screen";
                AutoRequest(NULL, &reqtext[0], NULL, &reqtext[1],
                            NULL, GADGETUP, 320, 60);
            }
            CloseLibrary((struct Library *) GfxBase);
        }
        CloseLibrary((struct Library *) IntuitionBase);
    }
}

/*
 * CheckPAL returns TRUE, if the the videomode of the specified public screen
 * (or default videmode) is PAL. If the screenname is NULL, the default public
 * screen will be used.
 */

BOOL CheckPAL(STRPTR screenname)
{
    struct Screen   *screen;
```

```
    ULONG           modeID = LORES_KEY;
    struct DisplayInfo displayinfo;
    BOOL            IsPAL;

    if (GfxBase->LibNode.lib_Version >= 36)
    {

        /*
         * We got V36, so lets use the new calls to find out what kind of
         * videomode the user (hopefully) prefers.
         */

        if (screen = LockPubScreen(screenname))
        {
            /*
             * Use graphics.library/GetVPModeID() to get the ModeID of the
             * specified screen. Will use the default public screen (Workbench
             * most of the time) if NULL It is _very_ unlikely that this would
             * be invalid, heck it's impossible.
             */
            if ((modeID = GetVPModeID(&(screen->ViewPort))) != INVALID_ID)
            {

                /*
                 * If the screen is in VGA mode, we can't tell whether the
                 * system is PAL or NTSC. So to be fullproof we fall back to
                 * the displayinfo of the default.monitor by inquiring about
                 * just the LORES_KEY displaymode if we don't know. The
                 * default.monitor reflects the initial video setup of the
                 * system, thus is an alias for either ntsc.monitor or
                 * pal.monitor. We only use the displaymode of the specified
                 * public screen if it's display mode is PAL or NTSC and NOT
                 * the default.
                 */

                if (!((modeID & MONITOR_ID_MASK) == NTSC_MONITOR_ID ||
                      (modeID & MONITOR_ID_MASK) == PAL_MONITOR_ID))
                    modeID = LORES_KEY;
            }
            UnlockPubScreen(NULL, screen);
        }                                       /* if fails modeID =
                                                 * LORES_KEY. Can't lock
                                                 * screen, so fall back on
                                                 * default monitor. */

        if (GetDisplayInfoData(NULL, (UBYTE *) & displayinfo,
                              sizeof(struct DisplayInfo), DTAG_DISP, modeID))
        {
            if (displayinfo.PropertyFlags & DIPF_IS_PAL)
                IsPAL = TRUE;
            else
                IsPAL = FALSE;                  /* Currently the default
                                                 * monitor is always either
                                                 * PAL or NTSC. */

        }
    }
    else                                        /* < V36. The enhancements to
                                                 * the videosystem in V36
                                                 * cannot be better expressed
                                                 * than with the simple way to
                                                 * determine PAL in V34. */
        IsPAL = (GfxBase->DisplayFlags & PAL) ? TRUE : FALSE;

    return (IsPAL);
}
```

Advanced Amiga Chip Set Graphics

by Allan Havemose

The capabilities of the Amiga chipsets have basically remained unchanged since the A1000. The ECS chipset now common in A500 and A2000 added viewport scrolling, superhires and VGA productivity modes, but didn't really add any capabilities to the standard PAL and NTSC modes.

The Advanced Amiga Chip set (AA) redefines graphics performance on the Amiga by adding many new features to existing modes and adding a palette of new modes. This talk focuses on the system software support for the AA chipset, explains the new functions, show how to use the display database to get characteristics about modes and discusses some of the caveat you will have to watch out for to be AA compatible.

Another design goal for AA graphics was to abstract some of the functionality away from the Amiga chipset. This is most evident in the functions for palette control and palette sharing. We have provided interfaces beyond the capabilities of the current chipset.

The AA Chip Set

Support up to 8 bitplanes in all modes.

8 bitplanes in non-HAM modes corresponds to 256 colors.
Enhanced HAM: 6 bit to select base register and 2 bit for mode control
HAM works in HIRES and SUPERHIRES.
Dual playfield with each playfield of up to 4 bitplanes

8 bits for Red, 8 bits for Green, and 8 bits for Blue. This translates to 16 million colors. There is 1 genlock bit.

2x and/or 4x bandwidth over ECS corresponding to 32 bit chip access and pagemode CHIP RAM.

New sprite features:

35ns, 70ns and 140 ns sprites independent of the view.
Attached sprites in all modes
Sprite width of 16,32 or 64 bits

Scan doubling: 15 KHz display and sprites can be promoted to 31 KHz. Interlaced 15KHz displays can be deinterlaced and displayed at 31 KHz.

Bitplane alignment issues: In order to use page mode access to Chip ram, it's required that each scan line is allocated on an 8 byte boundary for 4x modes and 4 byte boundary for 2x modes.

Palette Banking: The palette is arranged as 8 banks of 32 colors. Graphics can switch bank by writing to one RGA register.

Sprite Color Offsets: Sprite colors can now be chosen from the ranges 0-15, 16-31, ... (for even and odd sprites respectively)

AA graphics Software

AA graphics have spawned quite a substantial number of functions. The new functions serves two different purposes:

1. Support new AA chip features

2. Provide function calls for ''functions'' currently defined as macros. Using those functions will ensure easier migration to future platforms. Using the functions provides device-independent access to fields in the RastPort structure.

Get and Set Functions

```
ULONG GetAPen(rp)    Return current APen value
ULONG GetBPen(rp)    Return current BPen value
ULONG GetDrMd(rp)    Return current drawmode
ULONG GetOPen(rp)    Return current area outline pen
ULONG GetWrMsk(rp,m)    Return current Mask value
ULONG GetAfPt(w,&p,&n) Gets areafill pattern
VOID  GetCp_XY(rp,&x,&y)  Return current cursor position in rastport
VOID  SetAPen(rp,pen)    Sets APen
VOID  SetBPen(rp,pen)    Sets BPen
VOID  SetDrMd(rp,mode)   Sets DrawMode
VOID  SetWrMsk(rp,m)     Sets new Mask
VOID  SetOPen(rp,c)      Sets outline pen
VOID  SetAfPt(w,p,n)     Sets areafill pattern
VOID  BoundaryOff(rp)    Turn off area outlining. Was a macro.
VOID  SetABPenDrMd(rp,apen,bpen,drmd)
```

Sets the APen, BPen, and DrawMode for a rastport. Faster than separate calls.

```
VOID  SetRPAttr(rp,taglist)
```

Sets RastPort attributes based on a taglist. Useful for setting attributes beyond those known today.

Note that a few of the functions in the list already are in *graphics.library*. They are listed here for completeness.

Device Information Functions

Up until AA, it has been "safe" to make certain assumptions about the graphics hardware. As an example, many programs assume that RGB values are 4 bit each. In System 2.0, the Display Database was introduced. An application should query the display database for information like number of bits per gun in the palette, max depth of specific modes etc. It strongly recommended to use the display database extensively. This is the only way your application can adapt to different types of display devices "transparently".

```
CalcIVG(v,vp)
```

Calculate the number of display lines needed to execute all the copper instructions in a ViewPort, considering the FMode, BitPlaneUsage, and number of lines that have to be blank. This will mainly be used by Intuition, which was previously hard coding it's Inter-ScreenGap value. It could also be used by such programs as SHAM and Dynamic-HIRES to calculate the number of lines needed to execute its copper instructions in the middle of a display window.

Color Map Functions

The color palette in AA is different in a lot of ways from ECS:

It has 24 bits per entry, plus one bit to select transparency.

There are 256 of them, which is enough for many programs running on the same screen to share the palette.

A ColorMap is a structure used to store a variable number of pixel attributes which are used by the display hardware to determine how to display a pixel.

The most important pixel attributes (channels) are the RGB components of the color. All graphics devices support setting RGB color components. This does not imply that a display device has to be RGB mapped. It means that a graphics device must be capable of converting colors specified by RGB values to its internal representation. Since RGB is the most important attribute, there are a greater variety of methods for controlling it.

Other pixel attributes are supported, and may be controlled if the underlying hardware supports it.

The standard way of specifying a channel value is as a left justified 32 bit fraction. All the upper bits that the hardware supports will be stored and used. To identify the new 32 bit color map function, the functions have been named as with a '32' in the name (i.e. SetRGB32(...)). Note that the RGB values are left justified. That means that you can use the same algorithms to calculate in old style 4 bit RGB values and the new 8 bit values, provided that the new 32 bit color map functions are used. The *graphics.library* calculates in the new format internally, so if you use SetRGB4(), *graphics.library* will convert the RGB value to the new format. In 32 bit format, an RGB value of 0x80000000 represent 50% intensity and 0x40000000 represent 25%.

The best way to promote a 4 bit-per-gun RGB value to the new 32 bit format is to copy the value to all nibbles of the 32 bit value. For example, a value of 0x4 should be stored as 0x44444444 in 32 bit mode. That guarantees equal scaling and use of the full color range. Remember that 4 bit-per-gun RGB values are used in ECS and earlier Amiga chip sets.

Note that setting an attribute and then retrieving the value by the corresponding ''GET'' call may not return the exact value that was passed. A truncated version will be returned. This can be taken advantage of in dithering routines, for instance.

```
VOID SetRGB32CM(cm,n,r,g,b)
```

Sets a palette entry in a colormap. RGB are 32 bit left justified unsigned values. Other channels wil not be affected.

```
VOID SetCMChannel32(vp,n,channel-id,value)
ULONG GetCMChannel32(cm,n,chan-id)
```

Used for getting/setting other palette attributes beyond RGB. `CHAN_ID_ALPHA` sets the alpha channel.

```
VOID LoadRGB32(vp,firstcolor,colors,count)
```

Load up a whole set of RGB values into a color map. Other channels will not be affected. Colors are (32 bits)*(3 components) each.

```
VOID SetRGB32(vp,n,r,g,b)
```

Sets one palette entry to the specified color.

```
VOID GetRGB32(cm,first,n,&rgb)
```

Reads 'n' colors from the palette, starting at 'first'. There must be at least 3*n longwords of storage at 'RGB'.

```
pen_num=AllocBestColor(cm,R,G,B,precision,maxcolor,proc)
```

This function allows programs running in shared ViewPorts to manage their access to the color palette, by allocating entries in the palette for shared access. A reference count will be maintained for each color. The precision argument can be:

- `PRECISION_VERY_LOW` tells the palette map allocator to not allocate a new entry, regardless of the error magnitude.
- `PRECISION_LOW` settles for quite a discrepancy. Should be used for text colors, etc.
- `PRECISION_MED` medium color error. Should be used for GUI colors, icon colors, etc.
- `PRECISION_HIGH` small color error. Should be used for bitmapped images, etc.

The error metric used is based upon an Euclidian distance approximation, with the percentage of free color entries factored in.

If 'proc' is non-zero, then it is a pointer to a user routine which will be called for each color. Using a user comparison routine may be slower than the system's routine, because the default system routine may not examine all colors.

```
VOID FreeCMEntry(cm,n)
```

Frees a shared or unshared palette entry.

```
LONG AllocCMEntry(cm,n,r,g,b,flags)(a0,d0/d1/d2/d3/d4)
```

This allocates a palette entry. N specifies which entry, or -1 for the next available one. Flags may be set to `PAL_EXCLUSIVE` to allocate an entry for exclusive use, or 0 for shared use. The palette entry will be set to color R,G,B.

```
MatchColor(cm,r,g,b,c1,c2,fn)
```

Finds the closest matching color in a color map, between indexes c1 and c2. This function does not allocate colors or restrict itself to sharable colors. It exists for two reasons:

1. For desk-accessory type applications opening on public screens which want to use one of the application's colors, with no guarantee that it won't change from under them. This should be used as a fallback function for such programs when they find no sharable or allocatable colors available.

2. For mapping of colors to internal color palettes on screens without sharable colors (a 256 color DPaint comes to mind).

If `fn !=NULL` the user supplied function will be called. A value of zero means to use the default comparison function.

Bitmap Functions

These functions exist because the new AA chips have alignment restrictions in high bandwidth modes. Changing InitBitMap() and AllocRaster() to obey these restrictions appears to be very incompatible (AllocRaster() does not take a BitMap pointer for instance). Bitmaps created by AllocRaster() with a multiple of 32 or 64 pixels per line will be compatible with high fetch modes (2x or 4x respectively). Incompatible ones will fall back to 1x mode if 1x mode is capable of displaying the screen.

AllocBitMap() allocates an entire BitMap structure, and the display memory for it. AllocBitMap() allows you to use more than 8 planes, and also allows you to specify another bitmap pointer, thus telling the system to allocate the bitmap to be "like" another bitmap. A bitmap allocated in such a matter may be able to blit to this bitmap faster. Such a bitmap may be stored in a foreign device's local memory. Do not assume anything about the structure of a bitmap allocated in this manner.

The size of a bitmap structure is subject to change in future graphics releases. Thus, you should use AllocBitMap()/FreeBitMap() for your raster allocation.

```
bmptr=AllocBitMap(xsize,ysize,flags, depth, friend_bitmap)
VOID FreeBitMap(bmptr)
```

Available flags are:

`BMF_DISPLAYABLE` specifies that the memory allocated should satisfy the alignment restrictions for the display hardware. These restrictions are more strict then those for other bitmaps.

`BMF_CLEAR` tells graphics that the allocated memory should be cleared.

A bitmap created with `BMF_DISPLAYABLE` can be blitted as well as displayed, but one created without this flag may not be displayable. In other words, the alignment/address restrictions for blitable bitmaps are never more restrictive than those for displayable bitmaps.

Friend_bitmap specifies a bitmap to allocate this bitmap to be like. Passing `NULL` will get you one in the standard amiga format. The depth argument may be ignored when there is a friend_bitmap, if blits from different depth bitmaps are not efficient on the target graphics device. When allocating using a friend bitmap, it is not safe to assume anything about the structure of the bitmap data. The only safe operations to perform on it are:

1. blitting it to another bitmap, which must be either a standard Amiga bitmap or a friend of this bitmap,
2. blitting from this bitmap to a friend bitmap or to a standard amiga bitmap, and
3. attaching it to a rastport and making rendering calls. It is also safe to examine the Depth field of the bitmap to determine how deep of a bitmap must be allocated to hold a copy of this one. Good arguments to pass for the friend_bitmap are your window's RPort->BitMap, and your screen's RastPort->BitMap. *Do not pass &(screenptr->BitMap)!* Under RTG, this structure may not be valid on all devices.

Page Flipping Support

```
VOID ChangeVPBitMap(vp,bm,dbufstruct)

   struct DBufStruct {
        struct MinNode db_Node; /* do not mess with! */
        struct Task *db_SigTask;
        ULONG db_SigMask;  /* set of signals to send when OK.*/
        ULONG reserved[2];  /* for future use. set to 0! */
   }
```

Changes the bitmap for a ViewPort. Dbufstruct contains a signal mask which can be used to signal your task when the buffers have swapped. The bitmap swapped in *must* be of the same organization (BytesPerRow) and alignment as the one initially installed. If a dbufstruct pointer of `NULL` is passed, then no notification will be done.

Not all display modes or devices are capable of this call. You can check a flag in the display database to determine this.

Sprite Functions

Graphics sprite functions will be extended to understand large sprites, selectable sprite pixel resolution, and movement of scan-doubled sprites. Sprite positioning will no longer be rounded down to lo-res pixel resolution.

Sprite size is set on a per ViewPort basis. However, different sprite sizes on a per ViewPort basis are *not* compatible with Intuition because of the need to keep the sprites from overlapping other ViewPorts with different sprite size settings. Thus, Intuition should always set all its screens to have the same sprite size. The only reason not to use the largest size possible is compatibility. This will most likely have to be handled by Intuition and Preferences.

A game with custom ViewPorts could use different sprite sizes, as long as it keeps the sprites from crossing into ViewPorts with different sprite sizes. This might not be particularly useful, though.

The function:

```
    UWORD * BitMapToSprite(bm,SizeInWords,scalex, scaley)
```

Will convert a bitmap to a sprite. The BytesPerRow and height fields of the bitmap determine the source data size. Words beyond the specified horizontal size will be ignored. This data_ptr can be stuffed into the posctldata of a SimpleSprite structure. Scalex and Scaley allow integer scaling up or down by positive or negative powers of two.

```
    VOID FreeSpriteData(data_ptr)
```

MoveSprite() determines the format of the sprite data by the ViewPort's ColorMap. If MoveSprite() is called in View relative mode, then the ColorMap of the first ViewPort is examined (in view relative mode, it is assumed that all ViewPorts have the same sprite size set).

When `NULL` pointers or old-style colormaps are encountered, the sprite size will fall back to one-word-wide mode. The maximum sprite width possible may be determined from GfxBase->MaxSpriteWidth. Intuition may eventually be able to automatically selecting different sized pointers.

New Videocontrol Tags

`VTAG_SETSPRITEWIDTH` get/set the sprite width in words.

`VTAG_SETCOLORBASE` get/set the palette base for this ViewPort. Should be a multiple of the number of colors in the ViewPort, otherwise interesting effects may happen. Do not set the colorbase to a color greater than the number of colors in the color map!

`VTAG_SET_P2_OFS` get/set the offset of playfield 2 from playfield 1. *Must* be a power of two.

`VTAG_SET_SPR_OFS_EVEN` get/set the color offset for the even sprites. *Must* be multiple of 16.

`VTAG_SET_SPR_OFS_ODD` get/set the color offset for the odd sprites. *Must* be a multiple of 16.

`VTAG_BORDERSPRITE` get/set the enable for sprites in borders.

`VTAG_SPRITE_RES` get/set the sprite pixel size.

Using The Display Database

Using the display database to get information about display modes is the only way to make sure that your application gets valid information. Many of the assumptions that were true for pre AA chips are not valid anymore. Don't *ever* rely on specifics that can be empirically determined today - you can be sure that those values will change as the Amiga migrates to more powerful chip sets.

The display database is a part of the Amiga OS Release 2.0.

For information and example code on how to use the display database, have a look at the two *Amiga Mail* articles by Ewout Walraven, CATS:

''Introduction to V36 Screens & Windows'', Sep-Oct 90
''Opening Screens & Windows on any Amiga'', Jan-Feb 91

additions to the display database for AA:

1. DisplayInfo now have Red, Green and Blue bit fields.
2. New DIPF defines:

```
DIPF_SUPPORT_DBUF          /* supports double buffering (V39) */
DIPF_IS_SPRITES_ATT        /* supports attached sprites (V39) */
DIPF_IS_SPRITES_CHNG_RES /* supports variable sprite resolution (V39) */
DIPF_IS_SPRITES_BORDER     /* sprite can be displayed in the border (V39) */
```

3. To query number of differt displayable colors:

In release 2.0 use dinfo->PaletteRange
In release 3.0 use dinfo->RedBits
dinfo->BlueBits
dinfo->GreenBits

Compatibility Issues

Allocating BitMaps:

To use 2x and 4x modes, the bitmap data must be aligned on a 64 bit boundary. Use the new function AllocBitMap() instead of AllocRaster(). Using AllocBitMap() will ensure optimal alignment of bitmap data.

MakeVPort:

MakeVPort() now returns an error value. MakeVPort() can now fail if a high bandwidth mode is requested and the bitmap data isn't alligned properly. The error `MVP_NO_DISPLAY` is returned in such a case.

Sprites:

Wide sprites expose two different problems:

1. The data format is different for different modes: Use the new BitMapToSprite() function.
2. Moving a wide Intuition pointer into a ViewPort with old style (16 bit) sprites causes a garbeled pointer when entering the new ViewPort. Sprites can independent of the underlying View. Sprites can be moved in Lores, Hires, or SuperHires resolution. Remember that View resolution and Sprite resolution are independent.

Scrolling:

A View can be scrolled a full resolution, i.e. a Hires screen can be scrolled at Hires precision.

Copper lists:

The copper lists have changed for AA graphics. Poking copper lists is a sure way to crash the system.

High bandwidth modes and boot disk games:

Graphics boots up into ECS mode on a AA machine (for compatibility with games. i.e. avoid using higher bandwidths on autobooting games). A program in the *startup-sequence* enables the AA features. This should be run early in the *startup-sequence* before any screens have been opened (especially before *iprefs*).

* * *

Intuition V39 Documentation

by Peter Cherna

This document is confidential and proprietary, and subject to change.

The V39 Intuition update consists primarily of features to support the AA chip set and V39 *graphics.library* features. In addition, a small number of pure Intuition features have been added. By and large, we have concentrated on features that do not introduce compatibility burdens on the developer disproportionate to the benefit returned. For example, the requests for several of the V39 features are cleanly ignored by Release 2.0 (V36 or V37 ROM).

(NB: At the end of this document, there is a short section describing some of the more important changes made to Intuition for V37. These changes were not described at previous DevCons).

NewLook Menus and Screen Bar:

The menus and screen bar were one key visual element not revisited during NewLook development for V36/V37. In particular, the new color scheme now means relatively ugly grey on black menus. This has now been addressed by adding additional color control. At the same time, the Amiga-key and checkmark symbols are now scalable.

To receive the full-blown NewLook menus, the screen opener must be aware of new pens in the SA_Pens array, and the window opener must be aware of NewLook menus. Unaware windows behave entirely as before. Aware windows on unaware screens will get the Amiga-key and checkmark scaling in the traditional color scheme.

How to open a screen which has a NewLook screen bar and supports NewLook menus (this code also works perfectly under V36/V37):

```
UWORD mypens[] =
{
    .../* pens as before */

    1, /* BARDETAILPEN *//* NEW */
    2, /* BARBLOCKPEN *//* NEW */
    1,  /* BARTRIMPEN *//* NEW */
    ~0,/* Terminator, as before */
};

    myScreen = OpenScreenTags( &myNewScreen,
        ...,
        SA_Pens, mypens,
        ...,
        TAG_DONE );
```

The simplest way to open a window which will have NewLook menus is to use GadTools. Note that this code works perfectly under V36/V37):

```
myWindow = OpenWindowTags( &myNewWindow,
        ...,
        WA_NewLookMenus, TRUE,                          /* NEW */
        ...,
        TAG_DONE );

menu = CreateMenus( myNewMenu,
        ...,
        TAG_DONE );

LayoutMenus( menu,
        ...
        GTMN_NewLookMenus, TRUE,                        /* NEW */
        TAG_DONE );
```

New or changed tags to OpenScreenTags():

(changed) SA_Pens (UWORD *): There are now three new pens, BARDETAILPEN, BARBLOCKPEN, and BARTRIMPEN, used for rendering text/graphics, filled areas, and trim of the menus and title-bar, respectively. They default to the screen's detail, block, and block pens respectively. The intended use is to make BARDETAILPEN and BARTRIMPEN black and BARBLOCKPEN white. The screen's title-bar will be rendered out of these pens. These additional pens are ignored by V36 and V37.

New or changed tags to OpenWindowTags():

(new) WA_NewLookMenus (BOOL): Requests NewLook menu treatment (defaults to FALSE). Menu colors are derived from new screen pens, instead of window detail and block pens. Ignored by V36/V37.

(changed) WA_Checkmark (struct Image *): Image to use as a checkmark in menus. If WA_NewLookMenus is not specified, the default will be the traditional checkmark in the original colors. If you have requested WA_NewLookMenus, then the default will be an appropriately colored checkmark, scaled to the screen's font. Alternately, you can provide a custom one, which you can design yourself or get from sysiclass (see below - use this if your menu-font is different from the screen's font).

(new) WA_AmigaKey (struct Image *): Image to use as the Amiga-key symbol in menus. If WA_NewLookMenus is not specified, the default will be the traditional Amiga-key symbol in the original colors. If you've requested WA_NewLookMenus, then the default will be an appropriately colored Amiga-key, scaled to the screen's font. Alternately, you can provide a custom one, which you can design yourself or get from sysiclass (see below - use this if your menu-font is different from the screen's font). Ignored by V36/V37.

New or changed tags for sysiclass:

(new) SYSIA_ReferenceFont (struct TextFont *): Some sysiclass images can be scaled to a size appropriate for a particular font. Use this tag to specify the font to use. If omitted, the SYSIA_DrawInfo dri_Font will be used. Make no assumptions as to the size of the resulting image, but rather read its dimensions after creation (image->Width and image->Height).

Ignored by V36/V37.

```
/* Use this if your menu-items have a custom font different
 * from the screen's font (don't forget to check for failure,
 * and call DisposeObject() when done):
 */
check_image = NewObject(NULL, "sysiclass",
    SYSIA_DrawInfo, drawinfo,
    SYSIA_Which, MENUCHECK,
    SYSIA_ReferenceFont, customfont,
    TAG_DONE);

amigakey_image = NewObject(NULL, "sysiclass",
    SYSIA_DrawInfo, drawinfo,
    SYSIA_Which, AMIGAKEY,
    SYSIA_ReferenceFont, customfont,
    TAG_DONE);
```

Matching changes to GadTools:

New or changed tags for LayoutMenu{Item}s():

(new) GTMN_NewLookMenus (BOOL): If you ask for WA_NewLookMenus for your window, you should ask for this tag as well. This informs GadTools to use the appropriate checkmark, Amiga-key, and colors. Ignored by V36/V37.

(new) GTMN_Checkmark (struct Image *): If you are using a custom image for a checkmark (WA_Checkmark), also pass it to GadTools, so it can lay the menus out accordingly. Ignored by V36/V37.

(new) GTMN_AmigaKey (struct Image *): If you are using a custom image for the Amiga-key in menus (WA_AmigaKey), also pass it to GadTools, so it can lay the menus out accordingly. Ignored by V36/V37.

(new) GTMN_FrontPen (ULONG): This tag has existed for CreateMenus(), but now it's heeded by LayoutMenu{Item}s(), too. If a legitimate pen number is supplied, it is used for coloring the menu items (in preference to anything passed to CreateMenus()). If GTMN_NewLookMenus has been specified, this tag defaults to using the screen's BARDETAILPEN, else it defaults to ''do nothing''. Ignored by V36/V37.

NewLook menus without GadTools:

If you intend to lay out your own menus (instead of using GadTools) here is what you need to know. For windows that do not specify {WA_NewLookMenus,TRUE}, their menu's Amiga-key symbol and checkmark will default to a width of COMMWIDTH (or LOWCOMMWIDTH) and CHECKWIDTH (or LOWCHECKWIDTH) as before. For windows with NewLook menus, you may find the default Amiga-key image and checkmark image in the DrawInfo dri_CheckMark and dri_AmigaKey field, and read the image widths from there. These fields were NULL under V36 and V37. These images are scaled to match the screen's font. If you wish to use some other font for your menu-items, you may create suitable Amiga-key and checkmark images as shown above.

For NewLook menus, you should use the screen's DrawInfo BARDETAILPEN and BARBLOCKPEN for the imagery in your items.

AA Support Issues

Consult the graphics database to see what modes and depths are available. You can now pass new values to SA_Depth (up to depth 8) and SA_DisplayID.

Many other features are available through *graphics.library*/VideoControl(), SpriteControl(), etc.

Inter-screen gap is now variable per screen, instead of three non-lace lines.

At the time of writing, details for sprite-support had not been finalized. Thus, only a brief general discussion is provided.

The AA chipset offers sprites in low-speed, high-speed, or super-speed pixels (140 ns, 70 ns, or 35 ns pixels), and in 16, 32, and 64 bit widths. We intend to allow the Intuition pointer to take advantage of these features.

Graphics is providing sprite-control functions, as well as a function that accepts sprite-data in the form of a regular BitMap. Graphics will internally massage the BitMap into whatever private form the hardware needs. Intuition clients are expected to use the *graphics.library* conversion functions, and pass the processed form to Intuition.

There will be a new call akin to SetPointer() which permits specifying new modes.

There will be a system-default wait-pointer, which the user will be able to change through Preferences.

We are striving for high compatibility with applications that use existing sprite features under Intuition. It is essential that such applications call GetSprite() and FreeSprite() as appropriate.

Co-existence of new-mode sprites (other than the pointer) and Intuition is not assured, but is a goal. It will certainly be possible to bring up any sprites after ''taking over'' from Intuition.

AA Palette/Color Support

ColorMap:

Intuition will determine the appropriate sized ColorMap for your screen. It will contain (1<<depth) colors, but never less than 32. If you intend to use bitplane or sprite color-banking (which could increase the required depth), pass {SA_ColorMapEntries, depth} to override. SA_ColorMapEntries is safely ignored under V36/V37.

To enable features such as bitplane-banking and sprite-banking, etc., turn them on with graphics calls after opening your screen but prior to bringing it to the front.

Screen Colors:

The new SA_Colors16 tag accepts a ColorSpec structure where the color data is 16 bits-per-gun left-justified-truncated (SA_Colors took a ColorSpec with 6 bits-per-gun right-justified). Though both SA_Colors16 and SA_Colors may be specified, SA_Colors16 takes precedence. SA_Colors16 is completely ignored by V36/V37.

(NB: We may end up instead with SA_Colors32 and a new ColorMap32 structure. This would be more agreeable with the new *graphics.library*/SetRGB32(), but it does chew more space.)

Default colors beyond 32 come from *graphics.library*/GetColorMap(). The defaults for the first 32 colors are, as always:

- For all screens, colors 0-3 and 17-19 (sprites)) come from Preferences,
- For SA_FullPalette screens, colors 0-31 come from Preferences.

To set the colors of your screen once opened, use any of the old or new color setting calls from graphics, including SetRGB32(), SetRGB4(), LoadRGB4().

Palette-sharing:

The *graphics.library* now has functions for sharing colors on a screen. Color sharing allows multiple applications that co-exist on one screen to allocate colors for themselves, or to share similar colors with other users. In the graphics scheme, a color pen may be exclusive or shared. Shared pens are available to other applications needing a similar-enough color. Nobody may change the color of a shared pen. Exclusive pens are for the owner's use only, and may be changed at the owner's will. It is possible to allocate a color as absolute (by specific pen number) or not (any free pen will be used).

For public screens, the default will be:

Pens in SA_Pens: allocated as absolute-shared
All other pens: allocated as absolute-exclusive

The reasoning is thus:

Any pen given in SA_Pens is, by definition, shareable, since other applications are supposed to look there for pen information. Other pens are not shareable, and can legitimately be changed by the screen's owner (e.g., a paint program).

The new SA_NoAllocPalette tag allows an application to instruct Intuition to leave all pens other than those in SA_Pens unallocated. The use of this tag will be strongly encouraged. With SA_NoAllocPalette, the scheme becomes:

Pens in SA_Pens:	allocated as absolute-shared
All other pens:	ot allocated

The application may then allocate pens as needed. A paint package, for example, would allocate all colors it uses as exclusive. Other applications might allocate several colors as shared or exclusive. Since Intuition opens all public screens in private state, the application has a chance to allocate its colors before making the screen available to visitors (see SetPubScreenModes()).

For regular custom screens, there is no issue because only the application itself exists there. Since the arrangement chosen doesn't matter, the most forward-looking default is chosen, which is just like {SA_NoAllocPalette, TRUE}.

Pens in SA_Pens:	allocated as absolute-shared
All other pens:	not allocated

The situation on the Workbench screen is more complex, since it is highly desirable that there be free colors. Perhaps the best idea is to allocate SA_Pens as shared, and 0-3 as exclusive. Some consideration should possibly be given to ~0 to ~3.

New tag for OpenScreenTags():

> (new) SA_NoAllocPalette (BOOL): For public screens, controls which palette entries are allocated through the graphics AllocPalette() function. Intuition always allocates (as shared) those pens given in SA_Pens. If TRUE, asks Intuition to leave all other palette entries unallocated. If FALSE, then all other palette entries will be allocated as exclusive. Defaults to FALSE, for compatibility. We encourage public screen owners to use the graph!cs palette-management functions, and to set this attribute to TRUE. Attached Screens

It is becoming increasingly popular to have multiple screens open simultaneously. A typical use is an application that has a full-sized screen (perhaps in HAM-mode) as a canvas, and a short screen as a control panel or palette. Since it is desirable that these screens slide together, Intuition now provides the ability to attach screens together.

First, you must open the parent screen, which would typically be the canvas, main screen, highest on the display, or other large screen. child screens can then be attached to the parent screen through the use of a new tag.

Child screens are otherwise opened in the usual way, that is to say they don't inherit any properties from their parent, and they live on the regular screen list (IntuitionBase->FirstScreen->...). In particular this means you can run the same code under V36/V37 and you'll just lose the depth-arrangement and dragging relationships. That would yield the best situation under those versions, and no conditional code is required on your part.

Dragging Attached Screens:

Screens can move in two similar ways. First, any screen may be pulled up or down below its natural top edge, concealing or revealing screens that are behind it. Second, oversized screens may be moved in any direction to reveal hidden areas. Attached screens slide together when the parent screen is pulled up or down below its natural top edge, but they slide independently otherwise.

Moving a child screen does not affect the parent screen (or any other child screens), but does change its position relative to the parent. This allows the user or programmer free control to position things such as control panels. A child screen can never be moved higher than its parent.

It is quite possible and reasonable to have an oversized (scrolling) parent screen, with one or more child screens attached. Scrolling the previously hidden portions of the parent screen into view doesn't affect the position of the child screens.

Depth-Arranging Attached Screens:

Whenever the user depth-arranges any child or parent screen (i.e., with the depth gadget or by pressing Amiga-M or Amiga-N), that screen's whole family is depth-arranged as a unit. Likewise, if the programmer depth-arranges a parent screen, that screen's whole family moves as a unit. Since the depth gadgets of each member of a screen family are fully equivalent, we recommend that only one depth gadget exists among the members of a single screen family.

Example -- User presses Amiga-M:

(back)	Other	/ Child-3
	/ Child-3	Parent
	Parent	\ Child-2
	\ Child-2	\ Child-1
(front)	\ Child-1	Other

However, when the programmer depth-arranges a child screen (i.e., using ScreenToFront() or ScreenToBack()), that screen moves to the front or to the very back among the other screens in its family. It will not go behind (or in front) of another screen it wasn't already behind (or in front of). This allows the program to show or hide various toolbox screens.

Example -- ScreenToBack(Child-1):

(back)	Other	Other
	/ Child-3	/ Child-1
	Parent	/ Child-3
	\ Child-2	Parent
(front)	\ Child-1	\ Child-2

A parent screen may have several child screens. However, a child screen may not itself have children.

You must close any child screens before closing the parent screen.

New or changed tags to OpenScreenTags():

(new) SA_Attach (struct Screen *): Pointer to a parent screen that you wish to attach this screen to, for purposes of depth-arrangement and dragging (Ignored by V36/V37). Eliminate Intuition's use of AllocRemember()

Intuition no longer uses FreeRemember(FALSE) when windows and screens are opened. This should help reduce memory fragmentation somewhat.

Fix Depth-Gadget for Fully-Visible Windows

In V36/V37, when the single window depth gadget replaced the two gadgets that existed in V35 and earlier, its behavior was ''if the window is frontmost, send it to the back, otherwise send it to the front''. As well, shift-clicking on the depth gadget could be used to force a window to the back.

The confusing part is that it is possible to have a non-obscured window that is not frontmost, but clicking on its depth gadget appears to have no effect (it sends it to the front, but the visuals don't change). Such windows are now sent to the back as if they had been frontmost.

IDS_DISABLED Support for Gadget Images

V36 Intuition introduced custom images. Even though the custom image mechanism allowed for disabled image states, the gadget mechanism was still responsible for performing the ghosting. It is reasonable to ask Intuition to allow the image to perform its own ghosting, if it wanted something fancier than Intuition provides (for example, the ghosting pattern could be restricted to a non-rectangular area).

There is a new boolean image attribute, IA_SupportsDisable, which an imageclass should return if it knows how to handle its own disabled rendering. Under V39, whenever a gadget is added to a window (through a window-opening or gadget-adding call), Intuition will ask the gadget's image (via GetAttr()) about this new attribute. The new gadget flag GFLG_IMAGEDISABLE will be set accordingly (do *not* set this flag yourself!).

If this flag is set, then the gadget-rendering code will never render the ghosting pattern. Instead, the disabled-state is passed to the image class. (The image class will be called with the IDS_DISABLED state or the new IDS_SELECTEDDISABLED state, as appropriate). The Amiga User-Interface Style Guide recommends that the ghosting pattern be performed using the SHADOWPEN (see struct DrawInfo). The standard pattern for ghosting is as follows:

```
    UWORD ghost_pattern[2] =
    {
        0x4444,
        0x1111,
        };

    SetAfPt( rport, ghost_pattern, 1 );
    SetDrMd( rport, JAM1 );

  #define   IDS_SELECTEDDISABLED (8L)/* disabled and selected    */
```

System support for Double-Buffering

There is a new function in *graphics.library* that swaps the BitMap associated with a ViewPort at the appropriate time. It can signal you when the previous bitmap is detached.

There is an Intuition wrapper function that normally does the same thing, but when menus are up, or when a gadget is active, it fails immediately, and does not swap the bitmaps. That way, menus automatically stop the animation, which will resume as soon as the user lets go the menu button.

The new Intuition call is ChangeScreenBitMap(), which takes a screen pointer, bitmap pointer, and a graphics DBufStruct, and it invokes the underlying *graphics.library* function as appropriate.

During menu or gadget state, calls to ChangeScreenBitMap() won't take effect. The call will return immediately with a failure code.

This design is sufficient to make menus work, since menus always leave the bitmap unaltered.

For gadgets, there are basically three cases in double-buffered screens:

A) When you don't really need it - Intuition now allows screens to be attached, meaning you can put your gadgets into a control panel which is attached to a double-buffered screen that has no gadgets. (This even means that the animation can continue to roll while the user plays with the gadgets!) The problem is instantly solved by virtue of it never coming into being.

B) Gadgets whose imagery is unaltered when done - A number of people will want to double-buffer but have a screen drag-bar and depth-gadget. Now the depth-gadget (and many other gadgets) have the property that when the user is done with them, the pixels remain unaltered from the original case. So ignoring calls to ChangeScreenBitMap() means that such gadgets work, provided the application performs one preparatory step, namely that after opening the screen, it must copy the bits in the screen into the other bitmap(s). This way it doesn't matter which buffer is displayed when the gadget stuff happens, since the bits always come back to the same thing when the user is done.

C) Gadgets whose imagery is altered when done - Many kinds of gadget (eg. strings, prop gadgets, or toggling buttons) don't return to their initial visual state when the user is finished with them. It would be hard to support double-buffering screens that contain such gadgets, for synchronization reasons (the application may end up calling ChangeScreenBitMap() after the user has played with a gadget but before the application has become aware of that). It seems reasonable, therefore, to expect applications to put such gadgets in a different screen.

ModifyIDCMP() Failure Conditions Improved

The WindowPort and UserPort are the pair of message ports Intuition uses for communication to an application window. They are normally created at OpenWindow() time. However, when an application wants to share one UserPort across several windows, it will open its windows with a NULL IDCMP flag. The WindowPort is then only created at ModifyIDCMP() time. By creating the WindowPort at OpenWindow() time, we can virtually eliminate all possibility of failure at ModifyIDCMP() time, which, until V37, didn't have a return code. Thus, starting with V39, ModifyIDCMP() can't fail if the window already has a UserPort.

Summary:

V36 and earlier: ModifyIDCMP() might fail if the window is lacking a UserPort or WindowPort, and the new IDCMP-flags are non-NULL. There is no way to detect this failure. Other uses of ModifyIDCMP() cannot fail.

V37: ModifyIDCMP() can fail under the same conditions. You may check the return code to learn when this happens.

V39: ModifyIDCMP() might fail only if the window lacks a UserPort, and the new IDCMP-flags are non-NULL. You may check the return code to learn when this happens. Non-Draggable Screens

Under V34 and earlier, you could make a screen effectively non-draggable by obscuring its drag-bar with a window. Sometimes, a screen that was supposed to be draggable ended up non-draggable because the screen drag-bar got covered. In V36, Intuition introduced mouse-screen-drag, which is intended to allow any screen to be dragged, even when its drag-bar got accidentally covered or moved off-screen. Since accidentally and deliberately non-draggable screens are indistinguishable to Intuition, the ability to have non-draggable screens was lost.

The new SA_Draggable tag allows you to specify that a screen may not be dragged. This tag defaults to TRUE, but may be overridden as {SA_Draggable, FALSE}. Under earlier Kickstarts, this tag is ignored.

We strongly recommend that this tag only be used under special circumstances. The Amiga is a multi-tasking machine, and our users appreciate that. Non-draggable screens will impair the ability of users to control their multi-tasking environment. Valid uses may include screens with sprites or complex copper lists, and ''environments'' that legitimately want to dominate the display. IDCMP Notification When Windows Are Depth-Arranged

If the {WA_NotifyDepth,TRUE} tag is passed to OpenWindowTags(), then V39 Intuition will send an IDCMP_CHANGEWINDOW message when a window is depth-arranged. (Currently, IDCMP_CHANGEWINDOW is only sent when windows are moved and/or re-sized). You can recognize that a given IDCMP_CHANGEWINDOW event was caused by a depth-arrangement by looking at the IntuiMessage Code field. Look for values of CWCODE_MOVESIZE (0) (indicating a window that was moved and/or sized) or CWCODE_DEPTH (1) (indicating that a window was depth-arranged).

Note that IDCMP_CHANGEWINDOW is only sent when the window itself is operated on. If the window in question ends up frontmost by virtue of some other window being dismissed, no message is sent.

Under V36/V37, IDCMP_CHANGEWINDOW will only be sent for changes in the position or size of a window, and the IntuiMessage Code field will always be zero. Miscellaneous Bug Fixes

A few known Intuition bugs may be fixed. The following are known to be fixed in V39:

Clicking in the area of a screen that has no windows now selects that screen for the purposes of autoscrolling. (Under V36/V37, you needed to first activate a window on that screen).

The mouse-pointer used to sometimes vanish in the inter-screen gap when three or more screens were visible. V37 Changes

The following is a list of features changed or added for V37.

For V37, there is a new flag that can be set that makes a string gadget participate in <Tab> / <Shift>-<Tab> cyclic activation. If the user presses <Tab> in such a gadget, the next such gadget on the gadget list will be activated. Likewise, pressing <Shift>-<Tab> will activate the previous such gadget on the list. (Note that custom gadgets you design can join in this if they can be active in the sense that string gadgets can.)

Set GFLG_TABCYCLE in the gadget->Flags field of a string gadget or string-like custom gadget, or pass {GA_TabCycle, TRUE} when creating a boopsi string or string-like gadget.

Also new to V37, there is a flag you can set so that when a user presses the <Help> key while a string gadget is active, the gadget will end. You can check the IntuiMessage->Code field for the value 0x5F, which will let you know that <Help>, not <Return>, was pressed. This facilitates an application's handling of the <Help> key.

Under V37 and higher, pass the {WA_MenuHelp, TRUE} tag-item to OpenWindowTags() in order to enable the menu-help feature. With menu-help enabled, when the user presses the <Help> key on the keyboard during a menu session, Intuition will terminate the menu session and issue this event in place of an IDCMP_MENUPICK message.

Notes for menu-help:

- ***never*** follow the NextSelect link for MENUHELP messages.
- You will be able to hear MENUHELP for ghosted menus. (This lets you tell the user why the option is ghosted.)
- Be aware that you can receive a MENUHELP message whose code corresponds to a menu header or an item that has sub-items (this cannot happen for MENUPICK). The code may also be MENUNULL.

- LIMITATION: if the user extend-selects some checkmarked items with the mouse, then presses MENUHELP, your application will only hear the MENUHELP report. You must re-examine the state of your checkmarks when you get a MENUHELP.

Menu panels that extend beyond the limits of the screen are clipped to the screen limits, and don't crash the system anymore. This protection was added in late V36, so if you depend on it, you need to ensure you have V37.

Sub-menu panels no longer flash when you roll the mouse on or off them, or if you have gaps between the select-boxes of your subitems. (This also was added late in V36, so is only guaranteed there if you have V37).

Under V36, because of the processing needed for the new display modes, reading the mouse position from IntuitionBase->MouseX and MouseY or setting it with IECLASS_POINTERPOS on PAL machines was incompatible with V34. With V37, the V34-compatible behavior is restored.

To reduce confusion caused by stuffing NewWindow.IDCMPFlags values into the NewWindow.Flags field (and similar confusion with Gadget Flags, Activation, and Type), some familiar labels have been replaced by non-ambiguous ones, such as IDCMP_ACTIVEWINDOW, and WFLG_ACTIVATE. The old ones are still available in a new include file, *iobsolete.h*, which is automatically included.

Late in V37 development, it was discovered that the gadget Activation flag GACT_STRINGEXTEND was not properly ignored by V34. Therefore it is not safe to set this flag under V34. We felt such transparent compatibility was valuable, so we defined a new flag GFLG_STRINGEXTEND (belonging in the gadget Flags field) that is properly ignored by V34.

V37 also added a new tag and fixed an old tag for ''propgadclass'' boopsi gadgets. PGA_NewLook controls the PROPNEWLOOK flag, and PGA_Borderless now works correctly.

V37 Intuition also fixes the great majority of reported bugs and compatibility problems experienced with V36 Intuition.

◆

CDTV User Interface Guidelines

I. INTRODUCTION

II. USER INTERFACE GUIDELINES
  2.1 The Screen
  2.2 Remote Control and the User
  2.3 User Interaction
  2.4 Selection
  2.5 The Remote: Specification for Use with an Application
  2.6 Other Guidelines
  2.7 Open Issues

III. USER TRAINING OF TITLE USE
  3.1 Tutorial
  3.2 Help Function
  3.3 Error Messages
  3.4 Title Documentation for the User

IV. INTERNATIONAL CONSIDERATIONS
  4.1 Preparing Titles for Other Languages
    4.1.1 Allowing for PAL
    4.1.2 Language and Cultural Distinctions
  4.2 Foreign Language Conversion
    4.2.1 Segmentation of Audio Tracks to Allow Translation
    4.2.2 Appropriateness of Symbols (Icons)

APPENDICES
  A. The CDTV Network
    A.1 Commodore Support
    A.2 Reaching Other Developers
    A.3 Available Tools and Programs
  B. Glossary for Applications Use
  C. User Interface Design Considerations
  D. IR Remote Control Description
  E. User Interface Standards Index

I. Introduction to The User Interface Guidelines

The purpose of these guidelines is to promote a uniform look and feel to CDTV applications. What does "uniform look and feel" imply? As presented here, it means that many of the Amiga intuition capabilities are disallowed as being difficult to see in a living room environment or too complex or confusing to a new home user. It means that low level issues such as a small number of fonts and the colors to be used are strongly suggested. It means that many "high level" functions, such as control panels, list requesters, and scrolling functions are to be supplied by a toolkit (or descriptions provided prior to the toolkit being done.)

The effect of these standards will be to increase the uniformity of look and feel to the user across many different applications. Note that this should have no effect on the developer's freedom of information presentation or content (ie, an animation or graphic, which is what the user selected, will be under complete control of the developer). Also note that by supplying the toolkit pieces, the standard fonts, and other elements of the UI standard, the level of effort for developers should be greatly reduced.

The user interface for the CDTV device should reflect the intended user population and environment; that is, it should be aimed at use on televisions in a home recreational setting by unsophisticated and often brand new computer users. We are dealing with a TV user, not a PC user (i.e. entertainment not work), and should be motivated to build titles accordingly.

These user interface guidelines are presented to assist you in developing an application that is easy for the average CDTV owner to use. Remember that the CDTV player is not sold as a computer and that every time you deviate from the guidelines, you place another obstacle between your product and the user. Our competition is not other CDTV applications, it is television. With too many obstacles, the user will simply change the channel.

These User Interface guidelines are not laws set in stone.

This document should be considered a work in progress. We invite your questions and comments. If you would like to become more involved in the continuous process of CDTV UI standards evolution, contact Alan Campagna or Guy Wright, c/o CDTV, Commodore, West Chester, PA.

Use of This Document

This document, User Interface Guidelines, is written for CDTV developers to ensure that CDTV titles follow a consistent pattern for user interaction. At some point in the future, Commodore is likely to limit some developer benefits to titles that adhere to the standards. These might be inclusion in Commodore's booth at major exhibitions, representation on the Sampler disc, bundling opportunities, etc.

Equally as valuable as these pre-sales benefits are the post-sales value. Following the UI guidelines means minimized post-sales support. If a user learns how to operate one title that follows the guidelines, there will be hardly any learning curve for other titles. This not only means fewer support calls and better word of mouth PR, but is likely to result in better product reviews and fewer dissatisfied customers and returns.

It is planned to have the UI standards available in data base form. In the near future, it is planned to accompany this document with source code, IFF files, and working programs to illustrate the guidelines and demonstrate how they may be implemented.

Sections II, III and IV of this document describes the recommended guidelines. The philosophy of the guidelines, and the reasons behind these recommendations, is detailed in Appendix B.

Overview of CDTV UI Issues

The UI standards focus on issues for consideration by developers (especially Amiga developers) before and during the title design stage. Without keeping these standards in mind, a developer might have the tendency to design the title with a personal computer in mind and subsequently, at the detail level, turn to these UI notes for help - too late!

When starting a CDTV project, remember that good CDTV titles include the following characteristics:

1 Ease of use.
2 Consistency of operation. Uniform look and operation across all similar applications.
3 Minimum documentation, or intuitive operation in any situation.
4 Take advantage of the CDTV capabilities (using up to 650 Mbytes of data, including sound, music, and digitized speech, etc.)

II. User Interface Guidelines

This section presents the guidelines in the following order:

2.1 The Screen - The issues related to viewing distance, TV screens, colors, and fonts.

2.2 Remote control and the user - Use of the remote as a positioning device; limitations and suggested uses.

2.3 User Interaction - General and specific situations of user interaction with applications.

2.4 Selection - User interactions specific to selection situations.

2.5 The Remote: Specification for Use with an Application - Specific button usage.

2.6 Other Guidelines - Operation of the applications in booting up, exiting and other common situations.

2.7 Open issues - Items remaining to be specified.

2.1 The Screen

Televisions are not the same as computer RGB monitors. Television is an interlaced, overscanned medium. What looks good on a monitor, may look terrible on a TV set in the home. View your screens on a TV set (both PAL and NTSC) before you commit them to CDTV disc. No matter how good your application is, if it doesn't look good on the home TV set then the users will be disappointed.

Viewing Distance

Keep in mind that users will have to work with the application from a 6 to 8 foot viewing distance.

Simplicity of Screens

Screens must be simple. Colors must look good on TV. Fonts must be large. Symbols must be easily recognized; large in size, distinctive in color and shape, and meaningful to most viewers.

Fonts

All fonts should be anti-aliased, outlined or backgrounded.

Some suggested fonts: Helvetica, Diamond, Times. Stay above 20 point type.

Currently, two new fonts are available to CDTV developers from Commodore without charge or restriction.

Colors

Colors should be subdued rather than bright to minimize bleeding. On a color saturation scale of 0 to 15, colors should be no more than 12.

Background colors should be an off-white or grey. Avoid using pale colors (pale pink, green, etc) as these can give surprising results on NTSC or PAL. For example, an attractive pale, salmon pink on a PAL TV set becomes ugly virulent orange on an NTSC set. Note also that saturation levels vary from PAL to NTSC. Colors that appear ''normal'' in NTSC may wash out in PAL. Refer to Developer Notes section on PAL/NTSC considerations.

You should test your artwork on both PAL and NTSC TV sets, or monitors with composite input. Contact your local Commodore office for availability.

As a rule of thumb, it is preferable to have text in a dark color on top of a light background.

Color Dependence

Options should not be presented as distinct only in color. For example, do not say ''select the green box'' (eight percent of the male population is colorblind to at least one combination).

Flicker

Beware of stark contrasts (all-black text on an all-white background for example). These conditions may exascerbate the flicker on an interlaced screen.

In interlaced modes, flicker is more evident. Flickering is most notable over sharp contrasts. Horizontal lines flicker more than vertical or diagonal. PAL flickers more than NTSC. When using horizontal lines, make them at least 2 pixels wide.

To reduce flicker, you may use 'shadowed' fonts. This trick, often used on TV subtitling, is very effective.

Product Testing for TV Viewing

Perform final testing of products on a TV, not an RGB monitor. View your product from a normal TV viewing distance.

Number of Items on Screen

There should be no more than 9 symbols on screen at one time. There may be more items at one time if they are all of the same type (e.g., numbers in a list), and easily recognizable by the user.

Borders

Screen developers should work within a 'safe' area. When testing on a standard Amiga monitor, keep all vital information at least one inch (2.5 cm.) from all borders. This helps avoid problems with badly centered TV sets.

Note that screen borders may need to be changed when going from PAL to NTSC. PAL CDTV players display 256 lines (512 in interlaced mode); NTSC players display only 200 lines (400 interlaced).

Desktop Model

The home user is not in a productivity environment, and the TV screen is too small for intense scrutiny from 8 to 10 feet (2 to 3 meters) away. For these reasons, the following rules apply:

☐ No windows, no window borders - the whole screen is a window.
☐ No window gadgets; no windows to close or resize.
☐ No pull down menus. Use symbol menu trees. See Section 2.3.

Text

Wherever possible, *avoid text* for menu choices, in favor of symbols.

In text presentation, design the screen for large fonts. If the screen is static in nature (such as credits), use multiple screens or scrolling text if text does not fit well, rather than reducing point size.

If the application must present scrolling text to be read, make the scroll pleasant so that the large font size can be easily accomodated. If possible, propose various font sizes, and let the user choose his font of preference. This lets users choose a large font for working on TV sets, and a smaller one (displaying more text per screen) for working on a monitor.

Note that text by its very nature is language dependent, making the distribution of the application country dependent. If text is not necessary to the application, avoid it or localize it as much as possible.

Screen Saving

Since TV screen phosphor can be damaged by constant, unchanging pictures, Commodore strongly recommends that the developers provide for a 'time-out' on any screen. If user input has not been recieved for some period of time the screen will blank or go to a 'screen saver' mode (which can revert to the prior screen on any button event).

Use the BookIt utility in the startup-sequence to invoke the system time-out screen.

Avoid 'Dead Air'

Avoid black screens. One of the prime rules of TV broadcasting is to avoid ''dead air'', when nothing is being transmitted. This holds true for CDTV.

Avoid black, empty screens. When you must load a new image, either leave the previous image on screen, or add a message or symbol indicating action.

In order to avoid long bootup sequences, use the keeper utility in the startup-sequence to display something (a logo, perhaps) while loading an application.

2.2 Remote Control and the User

Primacy of the Remote

The standard interface shipped with the CDTV player is the infrared (IR) remote controller. All the features of an application should be accessable through the remote controller.

Applications *must* be designed to use the remote. Do not assume any other device. When the keyboard or joystick is required for the application, the remote should still function properly.

Most users will have only the remote. Design the application so that the remote can be used for everything.

Positioning; NO Screen Pointer

As a positioning device any remote unit will be awkward to use for fine adjustments; moreover, like a mouse, it is not completely intuitive. For that reason, the guidelines are to have:

☐ NO pointing of a cursor.
☐ NO gadgets for resizing, dragging etc.
☐ NO double-clicking.
☐ NO dragging.
☐ NO pull down menus.

This means simplifying screens, limiting the number of options on each screen, and supplying defaults wherever possible.

Key Functions

A set of definitions of key functions has been established. These are presented in section 2.5.

Mouse/Joystick Selection

If the operation of the application depends on this being in one state or the other, make sure the user is prompted to set it...try to determine that it is correct.

In general, do not depend on the position being correct, and adjust the software to operate well *in either position.*

2.3 User Interaction

The visual rules in 2.1 and the use of the remote in 2.2 are augmented here to describe the standard operation in dynamic or interactive situations.

The keys to the rules in this section are consistency and the use of visual and audio cues to better support user interaction.

Feedback

Both visual and audio feedback are *necessary.* When the user presses any button which can be interpreted by the application as meaning a change of state or user action request, *let the user know that the button was read.*

When the user has an item highlighted and presses one of the select keys on the IR remote, have a graphic indication that the program got the message. A "busy", "working", audio beep, screen flash, fade to black, or other visual/auditory signal will prevent users from pressing the select key over and over or thinking that the player is broken.

When the options are complex or detailed, provide help screens or audio help.

Negative Feedback. When the user presses buttons which are irrelevant to the process at hand, an audio cue that this is 'wrong', such as an unpleasant sound, would help greatly to channel the user in the right direction. If it is necessary to have buttons that are inoperable or irrelevant, those buttons should be ghosted and not selectable.

Disc Wait Feedback. Inform the user when disk I/O or processing is going on. Something like 'turning gears', 'Zzzzz', or 'Working...' message might be suitable. Try for animation wherever possible. Commodore will provide some default recommendations, but title designers may wish to use something more specific to their context.

Animating the book selected (such as making the book move out from the shelf) is a good way to both let the user know that the button select was seen, and keeping the users attention while the next screen is fetched and made ready for presentation.

Limitation of Activity

Maintain a single window of activity for selection operations, or ones which require the user to perform an operation. The Multi-window environment is too confusing for CDTV users. (Note that this does *not* mean that you should avoid multiple windows for presentation; these can be very interesting and useful.)

What should be avoided, is the necessity for the user to choose the process to which to send an input.

For example, a typical Amiga Workbench situation may have several active windows on the screen at the same time; the user indicates the window to which he/she wishes to input data by clicking on some portion of that window. This is what should be avoided in CDTV screens. Multiple requesters, such as those that appear in the CDTV ROM Preferences screen, are acceptable, since the user can clearly position to the requester desired with the arrow keys, so there is no confusion of where the next key strokes are going.

In some applications, such as games or those in which user input is treated as asynchronous with the application flow, care must be taken to keep the user from being confused as to what is expected and what effect pressing the button is having.

Consistency of Operation

Ensure that symbols (icons) have a standard meaning for button operation in similiar situations.

Section 2.5 presents standard uses of the buttons.

Menu Design

Selection situations should always look the same; the look may change, but feel and function are the same.

Use a symbol cycle, described below.

Use ghosting to indicate an option is not currently selectable.

Always provide a uniform exit symbol to return to upper most level of a menu tree (main menu).

Support a standard way for user to navigate menu trees.

Refer to Section 2.4 for more details on selection operation.

Symbol Cycle

The selection by the user of the next action to perform should be done by having the user cycle through a list of selectable options represented as symbols (subject to the limit of 9 symbols per screen). Always indicate the currently selected symbol. Outline, frame, flash, reverse the background color, or animate the items as they are highlighted. The user must be able to see which items are highlighted from across the room.

Provide symbols on screen to perform most operations. *Do not* replicate remote control buttons on screen as symbols.

The selected symbol is then chosen with the OK (enter) or A button.

Provide a simple cycle algorithm so that it is obvious how to get to any symbol quickly. Always provide a default symbol. Attempt to choose the most appropriate default. The default should be the symbol the user will most likely choose next.

Make the transition cycle smooth; control the 'bounce' so that the user does not have to hold the button too long, nor overshoot desired symbols.

Symbol Design

Make each option as clear as possible, from the design of the symbol.

The best symbol is clear and intuitive to the user - it does not have to be explained.

Symbols are better than text.

The average CDTV owner does not have the usual computer literate understanding of symbols and visual cues which are standard in the industry. Test your symbols on non-computer people first.

Use interesting symbols, specific to a situation, wherever possible. For instance, the 'bookshelf' selector is a good one because users can relate easily to 'choosing a book'; this gets them out of thinking of the machine as a computer and moves them into a familiar environment.

Help Function

Provide help at all levels of the application.

Build help into every menu selection.

Use the Escape '?' button to bring up a help screen.

The help screen can be used to allow the user to enter a tutorial on the product; this prevents the user from having to navigate back to the top (or wherever the tutorial entry point symbol was) to find out what to do.

The help screen can be used to provide an entrance to extended functionality, or special navigation of the tree, or setting of Preferences and modes. This allows the power user to get to things quickly, without bothering the novice user with extraneous decisions.

Escape Function

Use the Back button 'B' to allow the user to get out of a section of the menu, back up a level in a tree structure, or to stop a presentation and return to the menu which activated it. This is a very important 'user friendly' feature which most users will come to depend upon. It allows them to 'test' sections of the application without running the risk of having to sit through presentations which they don't want to see.

2.4 Selection

Selection situations are the most common application requirement. Users must chose areas of interest, actions to be performed, or set Preferences or options. Each situation is somewhat different; yet the similiarity of needs leads to the following general set of guidelines. Standardization of the selection rules will help all CDTV developers by giving CDTV users the confidence that they can pick up a new product and use it successfully without hours of training.

Selection Modes

The selection methods which can be used are, in order of preference;

❑ A menu of fixed items - This is by far the most useful form of selection. It may take the form of a tree of fixed menus, or a linear chain of connected menus.

❑ Lists of similiar items - Here the items represent similiar objects. These objects may be (for example) the countries of the world, or the food topics covered in a group of recipes. See below for a presentation of choice of lists versus menus.

❑ Browsing - Although this may be accomplished via menus or lists, browsing can also be supported through 'visual analogs', such as geographic movement, or simulation of motion through space.

Choice of Selection Mode

To chose between the use of the selection modes (menu networks, menu chains, or lists), the following guidelines are suggested. Each mode may be appropriate for a section of the application. The modes may be mixed, but the general order of use follows.

- ☐ Menu Networks - Use menu networks for top level choices, and for all action choices. Lists are seldom used for action choices. The Welcome Disc is a menu network. Most application selections can be handled with menu networks.
- ☐ Menu chains - Where a menu tree goes beyond 2 or 3 levels, it gets too complicated to remember where you are. To keep the number of levels small, break large numbers of choices into a menu chain, connecting each screen to the prior and the last with arrow symbols. At any level in a menu network, where the number of choices is larger than 9 items on a screen, make it a chain at that point. The linear form is better for equal level choices such as language Preferences. Tree form, which is most general, is preferable where a natural hierarchy exists.
- ☐ Lists - Lists are used mostly for selection of objects, not action. Each object bears a similiar relation to the others in the list (e.g., they are all planets, or bodies in the solar system). Mixing several types in a list is confusing and usually indicates that several lists should be used. The time, track, and language selections in the system audio control and preferences are examples of list selection.

The order of preference is therefore: menu network, menu chain, list.

Text Entry

Where absolutely necessary, text entry can be accomplished with the remote by: a) selection from a 'hot word' list, or making words in text already presented hot; b) presenting the user with a keyboard on the screen which treats the letters as selectable symbols. (The arcade name selection technique presents the letters as a linear, alphabetized list).

The discussion concerning the language dependence of any text-based application apply more so to text entry.

Guidelines for Menu Networks and Chains

Limit the number of symbols to 9 on screen at one time.

Provide feedback (see Section 2.3) for indicating selected symbol, and button presses.

Provide help on any screen: this may be a specific symbol, (standard help symbol is '?'); or use the Escape button.

Provide network control symbols: up to move to the last level, right to the next in the chain, left to move to the last in the chain. Audio forward-back button may also be used for flow.

This leaves about 4-6 'business' symbols on most screens, which is a determining factor in limitation of menu depth.

Selection of menu symbol: To select an item, activate an option or begin a task or function, the user will press the left select button marked 'A', or the Enter button. The button will be pressed once.

Navigation aids: in addition to the network control symbols, provide some form of 'map' for the user, letting them know where they are in the tree. This can be done by appropriate screen design, or titles or symbols which let them know where they are. The Help function can contain a brief description of where the user is.

Guidelines for Lists

Lists may take many forms. A list should have objects which are related to one another.

Scrolling is a natural function for lists. The nature of the scroll function is critical to the pleasant use of the list.

Refer to the ROM interface for examples of different types of lists:

☐ Numeric List, by digit - The time setting of the ROM Preferences shows how a number list can be scrolled by using the up/down arrows for list scroll, with the Enter button for selection of the item.

☐ Item List - The Track Select shows how items can be presented in a grid form, allowing the directional arrows to be used in both horizontal and vertical movement. In the case of the grid being larger than the screen, a directional arrow appears when the user positions the selection to the top or bottom rows; if there are more selections above or below, the arrow indicates that an arrow movement 'off' the grid will cause the grid to scroll. Refer to the ROM Audio panel for examples. Note that the Audio panel operation also illustrates the requirement for selection of multiple items from a list. This is covered below.

☐ List as a menu chain - Note that the Preferences operation of the ROM handles the selection of language as a fixed menu of items, chained together. It is worth noting that when the items of a list are fixed, as they are in this case, the use of a linear menu chain is much easier on the user and preferred to a general list format. Grid lists can be used where the selected items can be easily seen and recognized by the user. For example, numbers and single letters are 'recognized' rather than read. Grids are not appropriate for text, or symbols which are not easily recognized.

2.5 The Remote: Specification for Use With an Application

The use of the remote has been mentioned in the prior sections. Here we describe the use of each button, and additional features of the application which should be triggered directly from the remote. Two general rules should always be followed:

☐ **Observe standard uses of the keys.**

☐ **Do not duplicate key functions with symbols on the screen.**

'A' Button

This is the primary 'Enter' button. It activates selections in menus and lists. Since it is usually under the user's right thumb, it is easier to reach on the remote than the Enter button. It will be used almost exclusively for most applications.

Enter (OK) Button

In most situations it functions the same as 'A', but may have alternate meanings in multiple selection situations. Refer to the Audio Panel track selection operation for an example; the 'A' button means 'choose this item and add it to my list of selections', whereas the Enter button means 'I'm done with the list'.

The distinction between the use of 'A' and Enter is that 1) where two forms are needed, as in multiple selection, 'A' is the intermediate 'OK', and Enter is the final 'accept all' indicator; 2) in situations where data will be saved (eg, bookmarks or game scores saved to NVR) the Enter is required to force the user to certify that this is the action desired, since the 'A' button is too easy to hit accidentally.

'B' Button

This button functions as 'Exit this level', meaning 'Go up to last menu.'

'B' is used to break out of a section or level. It should also serve as an 'abort operation' button, to terminate an animation or presentation and return to the menu which caused the action to occur.

Escape Button

The Escape button should be treated as a "?" (question mark.) It's function in all cases should be to bring up some kind of help screen.

The help screen might be an access portal to many other features of the application; it could give the user the option of going through a demo or tutorial section, refer the user to their manual, bring up contextual help screens, or have special options such as 'return to top' 'return to previous screen', 'jump to another section', 'jump to another mode', etc. It could allow the user to change operational modes such as power user functions. This can be the primary means by which the user learns about the features of the application.

The distinction between the 'B' button and the Escape is that the 'B' is more simply a menu navigation and presentation control button, whereas the Escape (which is to be marked '?' on the remote) is to provide help and let the user know more about what to do.

A/B Buttons in Game Applications (ONLY):

They are the equivalent of the left and right mouse buttons in mouse mode. In joystick mode, only the left selector button will function as a fire button.

Note to game developers: the remapping of the remote buttons for special needs of games is certainly possible, but the standards for such uses are beyond the scope of these guidelines.

Arrow Buttons

On the left side of the remote is an eight-way direction key, which is used to move pointers on the screen, make selections, etc.

These arrow buttons are always used to move around. Move from symbol to symbol, move through a list, move to another selection, etc.

Joystick/Mouse Toggle Button

When in the joystick mode, the direction and fire button signals sent to the CDTV player will be in standard joy, four button, eight-way form. When the remote is in the mouse mode, intuition mouse movement calls will return values in four pixel increments rather than normal Amiga one-pixel increments. This was done, primarily to enhance the responsiveness and speed when moving a pointer. Developers should try and write their applications to trap for both modes if possible and notify the user to press the joy/mouse button to change modes if necesary. A simple "press left selection button to start" message to the user will let you determine what mode the remote is in. This button, currently a toggle button, will become a switch in the next revision of the remote control. The user will be able to identify the mode he is in by simply looking at the remote.

When in joystick mode, the only active buttons on the remote are the 'A' button and the four directional arrows. *All other buttons are disabled.* Should your applications need any other buttons, tell the user to switch to the MOUSE mode.

Numeric Buttons

Work the same as the numeric keypad on the Amiga keyboard.

Audio Control Buttons

Next to the volume controls are buttons for controlling all the audio functions of the CDTV player (rewind, fast-forward, play/pause, and stop).

In addition to audio control, the audio buttons on the remote control and the front panel of the player should be used as "navigational aids" for the user.

Play/Pause

A press of the play/pause button pauses the application. Another press re-starts the application from that point. Play/pause also acts as a mute button, halting the audio as well.

Fast Forward

The fast forward allows the user to skip forward through screens or a section.

Reverse

The reverse button allows the user to back up through screens or a section.

Stop

The stop button is used as a reset button to end an application, and re-start it from the beginning.

Obviously, there are some applications where use of the audio buttons for these purposes doesn't make sense, but in many cases the extra controls given to the user can simplify things quite a bit, since their use in the audio world is well understood.

GenLock Button

For use with the optional genlocking accessory. This button cycles the CDTV player through the three genlock modes: source only, mixed source and computer, and computer only.

TV/CDTV Button

For switching between CDTV and normal TV viewing when the CDTV player is connected to a TV via an RF modulator (this toggles the RF modulator and light on the player front panel).

Volume Buttons

These are next to the A/B selector keys. There are two buttons for adjusting the headphone level volume up and down. The volume level is indicated by the light on the front of the player.

Power Button

Turns the player on and off.

NOTE: if the player is turned off with the remote, it *must* be turned on with the remote, even if the panel on/off switch is used.

2.6 Other Guidelines

These issues relate to the operation of titles other than the interactive ones covered previously.

Documentation

There should be minimum printed documentation. A well designed application will result in intuitive operation in any situation.

Time-outs

Build "time-out" features into applications. When there is no input from the user after a few minutes, the program should revert to a self-running mode. This is particularly important at the beginning of a session. Startup screens should offer the choice of beginning right away or proceeding through a tutorial. After a minute or two without user response, jump right into the tutorial.

Use 'arcade style' demo of the product.

Refer to 'Screen Saving' in Section 2.1.

Exiting

This is a minor issue in most cases because the machine will re-boot when a new disc is inserted (unless the program has specifically requested an override of ~~the~~ re-boot function for multi-disc applications). However, all applications should exit cleanly and if any settings or Preferences have been altered, they must be reset before exiting.

An application should never return to the CLI window. If the application must exit, it should reset the system, and reboot.

Note that a program designer should 'trap' to prevent the user 'escaping through the top' of the menu tree by excessive 'B' button presses. A simple way to do this is to make the default on the top menu something other than the exit symbol, and to explicitly check that 'B' on the top menu is not accepted as a legal command.

Program Crashes

Certainly no developer will plan to ship a product with known bugs. However, some bugs do (rarely, we hope) go undetected until users do really unexpected things with the software.

Plan to have a mechanism to get information on what the user did to cause the problem.

In the case of system crashes, if applications use standard 1.3 operating system provided in the ROMs, the 'guru meditation' has been removed. In case of a fatal error, the system resets.

Autobooting

All applications must be auto-booting.

This can be simply accomplished by providing the usual startup.sequence file in the system S: directory. Do *not* hardcode your startup sequence to a specific device number.

The BookIt utility, available on the CDTV Tools diskette, should be included in your startup-sequence. It reads the user preference settings from the NVR, and centers the screen accordingly. BookIt can also be used to implement a screen saver.

Nationality and User Preferences Selection

Ideally, all applications will be either language independent or provide versions of the program in all languages. Developers should be aware that this machine will be sold in many countries and when the application does not currently support the language selected in Preferences, the user must be presented with alternatives.

While many people will adjust the Preferences (particularly nationality settings), most people will never touch them (if it ain't broke don't fix it). However, certain adjustments will have to be made by the user upon power up or after a loss of power.

One of the very first things that an application should check for is a match on language settings in Preferences. If the application does not provide a language specific version matching the particular language selected in Preferences then the application must provide the user with an alternate choice or set of choices.

2.7 Open Issues

Accessories

Design considerations for output devices.

☐ Printing - If an application requires printer control, developers should contact Commodore for printer drivers and Preferences code. There is a location in the NVR to store a printer selection, however, the printer drivers must be included in the title. When an invalid value is found in the printer save area in the NVR, the title should bring up a screen where the user can indicate the type of printer in use. This means that the first time a user selects ''print'', the title currently in use will put up a screen to set the printer Preferences.

☐ MIDI

☐ Personal Memory Cards

☐ Other I/O Concerns. To be addressed in the future.

III. User Training of Title Use

3.1 Tutorial

All applications should provide any necessary instructions on the disc itself. This can be done at the start if desired, but a means to escape the tutorial must be included. A link to the Help function should be provided at all times in the title.

3.2 Help Function

Applications should provide Help screens or audio help for the user. The Escape button has been designated as the 'Help' button (see Section 2.5).

When the application does not require Help screens, bring the instruction screens back up or deactivate the Help button function.

3.3 Error Messages

There will be a standardized set of error messages to communicate to the users when they encounter problems such as:Personal Memory Card full, file not found, Memory Card not present, printer trouble, etc.

This error functionality is to be provided as part of the developer support code to be distributed to registered CDTV developers.

3.4 Title Documentation For The User

Documention should be an integral part of the title itself. The best titles do not require hard copy documentation.

IV. International Considerations

It is important to keep in mind that much of Commodore's business is outside of the United States, particularly in western Europe. It is expected that a similar sales distribution of the CDTV player will occur.

Consequently, it is important to consider international aspects such as language, local customs and current success stories (i.e. what is ''in'' in the countries you are targeting?).Titles with worldwide appeal, which can be adapted easily to foreign environments, can expect far greater success than those developed exclusively for a single market.

4.1 Preparing Titles for Other Languages

4.1.1 Allowing for PAL

Refer to PAL/NTSC issues in Section 2.1.

☐ PAL has 512 lines in interlaced modes, NTSC has 400. This requires more memory and involves planning for adifferent aspect ratio.
☐ PAL is 50 Hz, NTSC is 60 Hz. This means that high resolution flickering is more prevalent especially if the contrasts are too high.
☐ A different color scheme may be required; PAL colors do not look the same as NTSC. Color saturation levels are particularly effected.

4.1.2 Language and Cultural Distinctions

☐ Many cultures are historically old and therefore rich in tradition; language is frequently a source of pride and great store is placed in proper use of the language. Do not assume that applications can be 'mechanically' translated into another culture just by looking up the words in a dictionary. In addition, pictures and symbols which are 'obvious' to one culture often make no sense (or worse, are insulting) to other cultures.
☐ All spoken words in an application need to be considered for translation to foreign languages.
☐ All titles should be tested with a native speaker who is CURRENTLY residing in the target market.

4.2 Foreign Language Conversion

4.2.1 Segmentation of audio tracks for translation

Segmentation of audio into pieces which can later be replaced with foreign language versions helps greatly in facilitiating the process later. This approach was taken in developing the Welcome Disc. Note that synchronization of translated tracks with graphics which have distinct time dependencies can cause some problems, since the translated narrative may take much more time than the original (e.g., English to German). In printed form, most European languages occupy 25-30 % more space than English (i.e. it takes more words to express the same thing).

4.2.2 Appropriateness of Symbols

To avoid translation of on screen text, the use of symbols is strongly recommended (refer to Section 2.1). Note the comment above as to cultural relativity;symbols may also require translation from one culture to another. The standard symbols (Section 2.3) should remain constant, but application specific symbols should be examined during the translation process.

4.2.3 Length of text

If you choose to use text for keywords in menus, remember that a word in English requiring 4 letters (such as EXIT) may require 6 or more in other languages (SORTIE in French, for example). This is another reason to consider using symbols as opposed to keywords in menus.

Appendix A: Glossary for Applications Use

The following glossary identifies terms which should be used by developers in building CDTV titles. These terms should be used as part of application text or narrative presentations, or in any documentation which refers to the subjects as noted.

It is presented alphabetically by term.Each term appears in the left column with its preliminary definition, when the term is thought appropriate to appear in user documentation.When the term has a note to use another word, this indicates that it should not be used, and the referenced term is to be used.

The glossary will be updated as necessary. It is available in numerous foreign languages.

'A' button	can also use ''select button'' or ''activate button'' .
ac powercord	use ''powercord'' .
accessories	CDTV player optional add-ons, not part of basic configuration (e.g. joystick, typing keyboard, etc.)
action	user does something.
activate	begin a non-interactive process such as starting a demo; (not to be used to indicate power/turning on).
animated	graphics displayed to simulate motion.
application	use ''title''.
arrow	directional indicators found on the remote control and the optional keyboard, use as ''up'',''down'', ''left'',''right''.
audio player	use ''cd-audio player''.
'B' button	the right select button on the remote control; used to return to, or ''back up'', to a previous choice or level.
begin	User starts an application. For audio, use ''play''.
bookmark	user opts to save some information, such as the status or high score of a game, or an application keeps track of an event, choice, or position.
boot	use ''start'' or ''restart''.

browse	to move in a non-linear fashion through a title.
button	used to activate; found on the remote control and other remote accessories, except for the optional typewriter-style or musical keyboard where the "buttons" are called "keys." To refer to a "button" that appears on the screen, use "symbol".
caddy	carrier or case into which cd is inserted to be played in the CDTV player.
cd	use "cd-audio" for audio discs, or "CDTV disc" for titles.
cd-audio disc	compact disc with audio only.
CDTV	Commodore Dynamic Total Vision. Use only as an adjective, not as a noun. "The CDTV player" is correct; "The CDTV" is not.
CDTV disc	compact disc specific to CDTV players.
CDTV player	the CDTV system.
channel	as in TV channel or audio channel (right and left stereo).
choice	an option available for user selection.
choose	use "select".
clear number	remove a number that the user entered.
clear screen	remove all user selections from the screen.
click	use "press" for keys, "push" for buttons.
compact disc	use "cd-audio disc" or "CDTV disc".
compatible	accessories that will work with CDTV players; other players that will run CDTV discs.
composite video	signals one type of CDTV player's video output.
composite video port	socket on back of CDTV player, into which a cable is inserted to connect a composite TV or monitor or VCR.
computer monitor	alternate to a TV for display device.

configuration	CDTV player and the accessories that are required for a particular application.
connect ... disconnect	to plug in ... unplug; applied only to hardware.
control panel	the plate on the front of the CDTV player, where the CD audio controls (Fast Forward, Play/Pause, etc.) are located.
cover	the front panel piece which is over the place to insert a personal memory card.
cursor	use ''pointer''.
cycle	loop through a variety of options.
data	uninterpreted elements of information.
default	pre-determined settings; user can override.
device	use ''accessory''.
direction arrow	the combination of left/right/up/down arrows on the remote control.
directory	avoid this term.
disc	proper spelling to be used when referring to a cd.i.e. ''disc'' is a read-only medium, whereas ''disk'' is a read-write medium.
disk	any read-write medium, such as a floppy or hard disk.
disk drive	mechanism for reading disks.
diskette	can also use ''floppy disk''
display panel	time/track indicators on the front of the CDTV player.
done	user has finished making a list or set of selections.
drawer	avoid this term.
eject	to make the caddy or personal memory card pop out.
elapsed time	the length of time passed since the start.
electronic keyboard	accessory involved in music; a piano-style keyboard.

enter	begin a user selected process or accept user input such as a number. (After a few months, remote controllers will be printed with "OK" instead of "ENTER")
enter button	avoid this term. This button should be referred to as the "OK button". (After a few months, remote controllers will be printed with "OK" in place of "ENTER")
escape	avoid this term.
Escape button	avoid this term. This button should be referred to as the Help button. (After a few months, remote controllers will be printed with "?" in place of "ESCAPE").
exit	to go back a level in a menu structure, or to leave a context.
external device	something attached to a CDTV player; an "accessory" (which is the preferred term).
fast forward	move quickly through a cd-audio or through a CDTV title.
file	organized collection of information.
fire button	button on a joystick, or the A button on the remote control, when in joystick mode; used to select a highlighted symbol or to "fire" in an arcade style game.
flash	flicker color of an object or light on and off or alternate different colors.
floppy disk	a read/write storage medium used in a floppy disk drive.
floppy disk drive	CDTV accessory that reads and writes floppy disks.
format	prepare a disk to have information stored on it.
genlock	a CDTV accessory which can display CDTV screens on top of video images.
graphics	information displayed pictorially.
guru	not relevant for CDTV players, as all errors should be trapped.
headphone	CDTV accessory to individually listen to audio.

help	detailed directions provided to user when requested; usually accessed via the Escape or ? button.
help button	button on the remote controller used to provide ''help'' information. The help button is currently printed as ''ESCAPE''. This will change within a few months to ''?''.
help screen	the information displayed by an application as a result of pressing the 'help' button.
highlight	to display selections and options in inverse video, or with a different border or other means of differentiating it.First the user ''highlights'' a choice with the arrow keys, then he ''selects'' it with the 'A' (or left select) button.
hit	use ''press'' for keys, ''push'' for buttons.
hook-up	use ''connect''
hot-link	means of connecting a word or phrase to all other relevant occurences of that word or phrase within an application.Can also use ''hot word''..
hypercard	avoid this term.
hyperlink	use ''hot-link''.
hypertext	use ''hot word''.
icon	use ''symbol''.
information	data put into a format to give it meaning.
infrared	the part of the light spectrum emitted by the remote control.Can be abbreviated as IR.
input	use ''action''.
insert	place the cd in its caddy or the caddy in the CDTV player, or the personal memory card in its slot.
instruction	explain usage to the user.
interactive	ability of user to influence or control what happens in a title.

inverse video	opposite colors such as negative and positive in the photographic sense; use ''highlight''.
IR	abbreviation for ''infrared''.
item	that which is selected in a menu or list; see also 'symbol'.
jack	a type of connection most often seen in audio.
jewel case	plastic box in which cd audio or CDTV discs may be shipped.
joy stick	CDTV accessory used as a remote control, usually for game playing; also a mode option for the CDTV standard remote control.
jump	go to another action, screen, symbol, etc.
key	''buttons'' on the optional typewriteror musical keyboard.
label	unique name.
landscape	horizontal format; graphic that is wider than it is tall.
left select button	''A'' button on the remote control; can also use ''select button''.
lesson	use ''instruction''.
light	indicates something; green light means power is on, amber light means disk is being accessed.
link	to cross-reference non-consecutive images or words.
loop	to go through a sequence and then repeatedly start again from the beginning.
main audio control	play/pause, forward/reverse, stop and volume controls on the front of the CDTV player(as opposed to those on the remote controller)
main power button	on/off button on the front of the CDTV player (as opposed to the one on the remote controller).
memory	use this term, not RAM.
menu	display of available choices.

MIDI	Musical Instrument Digital Interface; a standard means of connecting electronic musical instruments to one another, or to other devices, such as a CDTV player.
MIDI in ... out ... ports musical instruments.	socket on back of the CDTV player, used to connect electronic
modem	CDTV accessory enabling communication with another computer system via telephone.
monitor	alternative to TV for display of CDTV titles.
mouse	CDTV accessory enabling movement and selection of items on the screen.
NTSC	North American television standard. CDTV players work with both PAL and NTSC TV sets.Standard NTSC resolutions are 320x200; 320x400 (interlaced); 640x200; 640x400 (interlaced). Do not use this term in user documentation.
OK	button Button on the remote used to begin a user selected process or accept user input such as a number. (After a few months, remote controllers will be printed with "OK" in place of "ENTER").
output	any signal sent out of the CDTV player, such as the signal to a printer or midi instrument.
panel	use "control panel".
PAL	European television standard.CDTV players work with both PAL and NTSC TV sets. Standard PAL resolutions are 320x256; 320x512 (interlaced); 640x256; 640x512 (interlaced). Do not use this term in user documentation.
parallel printer	one type of printer that can be connected to the CDTV player.
pause	interrupt and hold at point of interrupt so that the application can be resumed from that point.
personal memory card	a battery backed-up memory card that can be inserted into a slot on the front of the CDTV player and used to store information.
picture	that which is displayed on the TV screen (e.g. clock picture).
play	begin/start cd-audio and, in some cases, a title.

player	use "cd-audio player" or "CDTV player"
pointer	indicates position on the TV; may be an arrow or other appropriate symbol.
ports	outlets on the CDTV player for connecting accessories.
powercord	cable to the source of power.
power button	on and off button on remote control or CDTV player.
preferences	specifies settings for clock, screen centering and national language.
press	applies to keys; "buttons" are "pushed", neither are "hit".
previous screen	the display immediately preceeding the current one.
printer	CDTV accessory.
program	use "application" or "title" (latter is preferred).
push	applies to buttons; "keys" are "pressed".Neither are "hit".
RAM	use "memory"
randomize button	allows audio tracks to be played in a nonconsecutive order; order cannot be user selected.
reboot	use "restart".
remote controller	device with buttons that send directions to the CDTV player via infrared signal.
remote	"other" buttonsbuttons on the remote other than the A and B buttons and the four arrow buttons e.g. "Help", "Play/Pause".
repeat	go back to beginning of process just completed and do it again.
reset	reestablish initial settings; restart a title.
restart	begin over again with initial settings.
retrieve	to bring back; to recall from memory or disc or personal memory card.

review	to browse available options such as a menu, or look over previous steps or selections.
rf converter	allows switching from CDTV to TV signals from remote control when TV is connected to the rf port of the player.
rf signals ... port	one of the video outputs of the CDTV player.
rgb signals ... port	one of the video outputs of the CDTV player.
s video signals ... port	one of the video outputs of the CDTV player; sometimes referred to as "Super VHS".
screen	that which is displayed on the TV.
screen blanker	use "screen saver".
screen saver	a function which avoids displaying a static image on a TV screen for long periods of time, to avoid phosphor burnout.
scroll	to move through a list on the screen.
search	to look for information based on a selected set of criteria.
select	to choose from available options. The user "highlights" his choice with the arrow keys, then "selects" with the "A" button.
select button	indicates or activates a chosen option; the left or "A" button on the remote.
self-running	non-interactive.
sequence	order, such as alphabetical or chronological
serial printer	one type of printer that can be connected to the CDTV player.
set (to)	to define with a specific value.
skip backward ... forward	means "pass over"; use "jump" for "go to another", or next action.
smart card	use "personal memory card".
software	use "title" (preferred) or "application".

start	begin
startup screen	initial CDTV display, before a disc is inserted.
stereo	refers to external sound system or audio output of CDTV player.
stop	to halt current function; opposite of activate.
switch	physical on/off toggle; or move to an alternative such as another screen.
symbol	graphic illustration of a choice which the user may select (e.g. track order symbol).
television	use "TV".
title	CDTV application.
toggle	alternate between two positions or options.
track	section on a cd-audio disc; each track contains a song or some other sound.
track segment	part of a track.
trackball controller	remote device with a ball which when rolled with the hand, moves pointer around the screen.
tutorial	use "instruction".
TV	television; acts as a display device for the CDTV player.
TV screen	portion of the TV where CDTV information is displayed.
user	interactive participant with the CDTV player.
volume	refers only to audio; increase and decrease.
wired mouse	use "mouse".

Appendix B: User Interface Design Considerations
CDTV Player Is Not a Computer

By Guy Wright

At the recent Amiga Developers conference in Atlanta in the spring of 1990, a number of CDTV seminars and forums were offered, and all of them were very well attended. I had the opportunity to sit in on all of them, and I moderated the CDTV open forum on Thursday evening. A few of the questions and comments stuck in my mind, and I thought that it might be a good idea to try and clarify a few points for everyone who could not attend the siminars.

The Number One question or misconception about CDTV players that I heard from many Amiga developers centered around one point:the notion that a CDTV player is just a modified Amiga with a CD-ROM drive. Let me say at the top that a CDTV player is *not* just a computer with a CD-ROM drive.

Although a CDTV player has the Amiga chip set inside and much of the functionality is based on Amiga technology, it is very important for developers to keep in mind that CDTV players will *not* be sold as a computers. The people who will be using CDTV players will probably not be computer literate, nor will they be willing to learn a lot of computer concepts. If they wanted a computer, they would buy a computer, not a CDTV device.

One of the central ideas behind the CDTV device is that it will be used in a living room environment by noncomputer people for noncomputer activities. It is not a Commodore oversight that the device won't have a keyboard or mouse shipped with it. This is done intentionally. We are trying to appeal to people who get easily confused by computer jargon. They may be techno-curious types, but for the most part they are buying an entertainment, educational, reference machine.

To put that in real terms that will be meaningful to developers, let's look at some of the user interface and environmental considerations.

Viewing Distance

The CDTV player will probably end up on top of the living room VCR. The device will be feeding video to a normal television set (NOT an RGB monitor). People will put in a CDTV disc, walk back to their easy chair or couch, pick up the IR remote controller, and begin. This means that they will be too far away to read small text on the screen. Normal Amiga fonts are far too small. Normal Amiga symbols (icons) are far too small. Anything at the edges, top, or bottom of the screen may not be visible depending on how well their sets are adjusted. Most colors will be poorly adjusted, so greens and blues may look the same.

All these factors mean that screens will have to be fairly simple. Use large fonts whenever text is to be displayed on the screen. You should test all of your screens on a TV set to get an idea how everything will look. Don't have small items on the screen that are critical to operation of the application. This means that you shouldn't present too many options at once or make them too small. The ''nine items on a screen'' suggested limit was arrived at because when you try to put too many items on a TV screen, they begin to get very small and difficult to see. Try not to make options color dependant. When you ask the user to select the blue symbol, he may not be able to tell which one is blue.

IR Remote

While there will be a few users out there who will buy the optional keyboard and mice, *Most* users will be using the infrared remote controller exclusively. You should be sure to design your application so that people can use the IR device for just about everything. This will probably mean simplifying screens, limiting the number of options on each screen, and supplying defaults wherever possible. During product development it is probably a good idea to ignore the mouse entirely. See Section 2.5 for a description of the remote controller.

The biggest hurdle to overcome when designing for the remote is positioning pointers. The remote is like a crude mouse, at best. Since it only registers in four pixel increments, fine positioning will be difficult. Wherever possible, you should design your application so that the user can cycle through the various options. Rather than saying ''click on this'' or ''point and click at that'', the user should be able to keep pressing the direction key on the IR remote until the option they desire is highlighted, and then they can press one of the select keys.

Users will not be able to drag items easily. They will not be able to move the cursor around the screen easily. Pointing at a small object on the screen will be difficult. Pull down menus, double-clicking, gadgets, etc., will either be impossible or difficult with the IR remote.

If you wish the user to make a choice between items or options, then build your program so that pressing the arrow keys (direction key on the IR remote) will cycle from one item to the next. ''Point and click'' operations translate into ''cycle through options until the desired option is highlighted, then press the A key.'' You should also be very clear about which items are highlighted as the user cycles through them. Outline, frame, flash, reverse the background color, or animate the items as they are highlighted. The users must be able to see which items are highlighted from across the room.

You should try and make each of the options as clear as possible (or supply help screens or audio help). Symbols are better than text. Digitized symbols are better than hand-created symbols. Animated symbols are better than static symbols. If you find that you have to explain to someone what an symbol means, it would be a good idea to rethink them. If the purpose of an symbol is clear and intuitive to the user, then it saves them frustration. It saves you time and money spent on user instructions as well. It is important to keep in mind that as a developer of computer products you have come to accept certain symbols and actions as intuitive. This may not be the case with an average CDTV owner. A white rectangle with a folded corner may symbolize a test/document file to most computer literate people, but it doesn't mean anything to a noncomputer person. Test your symbols on non-computer people first.

When the user has an item highlighted and presses one of the select keys on the remote, it is a good idea to have an indication that the program got the message. A ''busy'', ''working'', audio beep, screen flash, fade to black, or other visual/auditory signal will prevent users from pressing the select key over and over or thinking that the machine is broken. Most appliances in the home give almost instant feedback. They either start right away or at least beep to indicate that they know a button has been pressed.

The Passive User

Even though a CDTV player is an interactive multi-media device, most people still would like to sit back and have things handed to them. Given a choice between reading all about a subject or watching a TV show that only skims over the information, most people will watch the TV. Unless the user feels that the goal is valuable to them, they won't invest much time or effort getting it. They also don't want to read a lot of instructions or spend too much time learning a new system. You have to determine how much involvement to expect from the users and hopefully come to a balance. The more that a user wants to get something from your product, the more he will be willing to read, learn, and interact with it. If the information or reward is great enough, people will go to great lenghts to get it (just look at MS-DOS and how much effort it takes before someone reaps the benefits).

If the user feels only a casual interest, then the more obstacles you put in his way, the more frustrated the user will feel. In the case of the CDTV player an obstacle might be a long book of instructions, special keys to memorize, unique actions to perform, etc. Forcing the user to draw his chair closer to the screen in order to read or see something small is an obstacle. forcing him to do a lot of fine adjustments with the remote (such as fine positioning a pointer in a small area) is an obstacle. Perhaps the biggest obstacle would be requiring him to buy or have a keyboard or mouse.

Compare how difficult it is learning to drive a car with using a crepe maker. Most people will spend months learning to drive, stand in lines and take tests to get their license and spend thousands of dollars on cars because they see that the rewards are great. But even if you were given a crepe maker as a present, it ends up in a closet because it is just too much of an annoyance to make batter and clean the thing. If you feel that your particular application will be important enough to the user, you can make it as difficult as you like. If, on the other hand, you are offering them a crepe maker, then it better be self-cleaning.

You should try to build ''time-out'' features into your applications. In other words, if there is no input from the user after a few minutes, the program should revert to a self-running mode. This is particularly important at the beginning of a session. It would probably be a good idea to have your startup screens offer the coice of beginning right away or going through a turtorial. If nothing happens after a minute or two, jump right into the tutorial. Look at the way arcade game machines function. A mixture of self-running demos and instruction screens are constantly moving on the screen until someone puts in a quarter. People would much rather sit back and be shown how to use a thing (a few times if necessary) before they jump in rather than have to read printed instructions and then be dumped into something unfamiliar.

Don't be afraid to steal ideas from household appliances. How do people program a microwave oven? A stereo? A VCR? (Keep in mind that all over the world there are VCRs flashing 12:00...12:00...12:00 because setting the clock is just too much trouble or is too confusing.) How do people use a remote control device to watch TV? Do most people enter a channel number on the keypad, or isn't it just easier to press the up or down buttons a few times?

Summary: Design Considerations Unique to CDTV Applications.

Television (TV)

Televisions are not the same as computer RGB monitors. Television is an interlaced, overscanned medium. What looks good on a monitor, may look terrible on a TV set in the home. Some colors bleed, otheres do not stay 'true'. View your screens on a TV set before you commit them to CDTV disc. No matter how good your application is, if it doesn't look good on the home TV set then the users will be disappointed.

Use large fonts whenever text is to be displayed on the screen. Do not use small items on the screen that are critical to the operation of the application or present too many options at once. The suggested limit is nine items on a screen because when you put too many items on the TV, they get very small and difficult to see. Options that are color dependant are confusing and the user may not be able to tell which symbol is blue, preventing the proper selection. Test all of your screens on a TV to get an accurate impression of how it looks.

TV Imagery

Issues related to 6 to 8 foot viewing distance and TV resolution...

Fonts

Some suggested fonts: Helvetica, Diamond, Times, but just about any font will work if it isn't too fancy in the first place. In completely unscientific tests anything below 20 point type becomes difficult to read from more than ten feet. Also, broadcast television character generators almost always use anti-aliased fonts with a neutral colored outline. Ideally, the outline color is halfway between the character color and the background color. Outlines, borders, and drop-shadows greatly improve readability. Pastel colored fonts work better than bright or primary colors. Off-white works better than pure white. Yellows, greys, and pale blues seem to work best.)

NOTES: We are commissioning a few sets of fonts in various point sizes. They will be designed specifically for the CDTV support software and should be freely available to registered developers.

Colors

Colors should be subdued rather than bright (bright colors tend to 'bleed' on poorly adjusted TV sets.) If you are using a paint program like Deluxe Paint from Electronic Arts that have color values in a range from 0 to 15 we strongly recommend that no value go above 12 or 13. The only safe way to test colors and color combinations for display on an NTSC or PAL TV is with a waveform monitor. Maximum values 85% IRE. The best advice is to keep the color values below saturation and look at the results on a television set.

Nationality and User Preferences Selection

While many people will adjust the Prefs (particularly nationality settings), most people will never touch them (if it ain't broke don't fix it). However, certain adjustments will have to be made by the user upon power up or after a loss of power. We will try to make the transition between Preferences selection and normal use as seamless as possible.

Television Sizes

Some people will be using small TV sets. While we might have colored borders around the edges of screen developers should work within a 'safe' area. We will be supplying specific details on this 'safe' area dimensions, in the meanwhile allow at least an inch on all borders (assuming you are using a standard Amiga monitor).

Appendix C: IR Remote Control Description

Refer to the Developer Notes for a complete description of the key codes generated by the Remote Control.

On the left side of the remote is an eight-way direction key, which is used to move pointers on the screen, make selections, etc. There is another button on the remote for toggling how the remote direction key operates, either as a joystick or a mouse. when in the joystick mode, the direction and fire button signals sent to the player will be in standard joy, four button, eight-way form. When the remote is in the mouse mode, intuition mouse movement calls will return values in four pixel increments rather than normal Amiga one-pixel increments. This was done, primarily to enhance the responsiveness and speed when moving a pointer. Developers should try and write their applications to trap for both modes if possible and notify the user to press the joy/mouse button to change modes if necesary. A simple ''press left selection button to start'' message to the user will let you determine what mode the remote is in.

On the right side of the remote are left and right selector keys which are the equivalent of the left and right mouse buttons in mouse mode. In joystick mode, only the left selector button will function as a fire button.

Next to the selector keys are two buttons for adjusting the headphone volume up and down and a power button for turning the player on and off. Next to the volume controls are buttons for controlling all the audio functions of the CDTV player (rewind, fast-forward, play/pause, and stop).

Above the audio control buttons are three buttons, first a genlock button for use with the optional genlocking accessory. This button will cycle the CDTV player through the three genlock modes: source only, mixed source and computer, and computer only. Next is a button for switching between CDTV and normal TV viewing (this toggles the RF modulator and light on the player front panel). Finally is the joystick/mouse toggle button.

To the right of the direction key are 12 buttons (10 buttons numbered zero through nine, an escape button, and an enter button). They will be laid out in the same fashion as a telephone keypad excepting that the escape button is to the right of key 3, the zero button to the right of the 6, and the enter button to the right of the 9. These keys will send the same values as those found on the Amiga numeric keypad.

As a positioning device any remote unit will be awkward to use for fine adjustments (even a mouse is not completely intuitive). For that reason we should not have gadgets for resizing, dragging etc.

To avoid forcing the users to precisely position a pointer (and going along with some of the TV specific imagery rules) we felt it would be better to cycle through any list of selectable options.

We have established a set definition of any key functions (such as help or return to main screen). More thinking is required on how a user will use the IR device for scrolling through lists, controlling text speed and size, etc.. There may come a time when an application requires some form of text entry and not everyone will own a keyboard.

Escape button will become a "?" (question mark.) It's function in all cases is to bring up some kind of help screen. This help screen could be as simple as an symbol telling the user to put in the welcome disc (this would be the only thing handled by ROM.) Developers could have the ? button activate a generic help screen of their own, give the user the option of going through a demo or tutorial section, refer the user to their manual, bring up contextual help screens, or have special options such as 'return to top' 'return to previous screen', 'jump to another section', 'jump to another mode', etc..

The Return button will become an "OK" button. OK is used to make selections or activate an option in just about every case. In some cases (title dependant) the select or A/B buttons might perform this function for example if a title is Amiga mouse dependant (which we do not encourage.)

The arrow buttons are always used to move around. Move from symbol to symbol, move through a list, move to another selection, etc.

∞

Amiga AppShell

by David Junod

The purpose of this document is to describe, in technical detail, the construction of an AppShell application.

Scope

This document was written to provide the following information:

- Overview of the AppShell Components
- Technical Breakdown of the Components
- Overview of the Standard Message Handlers
- Implementation of a Custom Message Handler

Overview of the AppShell Components

The Amiga AppShell provides the developer of Amiga applications with the ability to easily incorporate a standard, consistent interface for the user.

The AppShell operates under the theory that applications can be broken into several separable modules:

User Interface - The means by which the user controls an application.
Application Functions - Functions specific to the application.
Standard Functions - Functions that are common to almost all applications, to provide a consistent, solid set of standard operations.

Of the three, the developer still has to define the first two. But the AppShell provides the third and makes the other two much easier.

Note: Unless otherwise stated, all references to the word "function" refers to either an Application Function or a Standard Function.

The following is a basic breakdown of the major components of the AppShell.

- open required libraries
- startup parameter parsing
    - locate configuration files
    - Workbench
    - Shell
- initialize message handlers (honor preferences)
    - ARexx
    - Command Shell
    - IDCMP (Intuition and GadTools)
    - Simple IPC
    - Asynchronous Tools
    - Workbench messages (AppIcon, AppMenu & AppWindow)
    - Others...
- handle messages (event processor) until quit
- shut down message handlers
- close libraries

The AppShell is implemented as a shared system library.

User Interface Description

The AppShell relies heavily on tags and structure arrays to describe the components of the user interface. All tag and structure arrays are unmodified by the AppShell. This allows any tag array to be used by different processes or be multi-threaded. Similarly, the entire AppShell is reentrant.

Note that not all of the data elements are copied. An effort has been made to indicate whether an element is copied or just its pointer is used.

Function Table

Every AppShell application has a function table. This table contains information on all the application functions, as well as the functions provided by each message handler. Every command to the application is dispatched by the AppShell function dispatcher to the correct function.

Each message handler can add functions to the function table. This ensures that each application has a standard base of functions.

Each function has a short name, which is used to bind the function to any particular user interface. A function could be bound to a menu item, a gadget, invoked from a script, or any other user interface.

Functions can have two different states:

Disabled - The function is not able to be invoked. Causes the dispatcher to return a failure-level error.

Enabled - The function is capable of being invoked. The default state of a function.

A function is disabled by sending the DISABLE command, followed by the function name:

```
DISABLE <function>
```

A function is enabled by sending the ENABLE command:

```
ENABLE <function>
```

If the function is bound to any particular graphical user interface item, then that item is also visually disabled. For example, a menu or gadget would be ghosted.

The ENABLE and DISABLE commands support wildcarding. For example, to disable all functions in your application until the user selects NEW, you could perform the following two commands:

```
DISABLE #?
ENABLE NEW ABOUT QUIT PREF#?
```

Names

Just about every AppShell item (functions, message handlers, and message handler objects) has a unique name. This allows the application to specify what is to be manipulated.

Preferences

User definable attributes of an application are called preferences. The AppShell provides the mechanism to read user preferences and transform the objects of the user interface to comply with those preferences. For example, if there is a GUI present in your application, then the AppShell allows the user to specify the screen mode and color palette (as well as a number of other attributes).

The Workbench preference editors are used whenever appropriate and therefore, their names are used also. For a list of the Workbench preference formats that the AppShell supports, see the IDCMP Message Handler section.

If the user wants to use different preferences for any particular application, then he would place the altered file into the application's preference directory. Permanent preference files would be placed in a directory named after the base name of the application in the ENVARC: directory.

The AppShell supports file notification on the preference files, so when a preference is modified by the appropriate Preference Editor, then the application's preferences are updated.

Preferences also involves localization. The AppShell has been designed to make use of the localization library when it becomes available.

Technical Breakdown Of Appshell Components

Application Variables

The AppShell maintains a structure which contains common variables required in just about all applications. This structure is named AppInfo and contains the following fields.

```
    /* AppInfo structure that contains an AppShell application's global
     * variables.  This structure is variable length, so NEVER embedd this
     * within your own structures.  If you must, then reference it by a
     * pointer. */
    struct AppInfo
    {
/* control information */
UBYTE *ai_TextRtn;              /* Text return string */
        LONG ai_Pri_Ret;                /* Primary error (severity) */
        LONG ai_Sec_Ret;                /* Secondary error (actual) */
        BOOL ai_Done;                   /* Done with main loop? */

        /* startup arguments */
        UWORD ai_Startup;               /* see defines below */
        LONG *ai_Options;               /* Option bucket */
        LONG ai_NumOpts;                /* Number of options */
        struct RDArgs *ai_ArgsPtr;      /* ReadArgs pointer */
        STRPTR *ai_FileArray;           /* MultiArg array (from the 0 bucket) */
                                        /* If the template specifies FILES/M, then
                                         * the array is expanded using pattern
                                         * matching and the resulting files are
                                         * added to the project list */
        struct DiskObject *ai_ProgDO;   /* Tool icon */

        /* base application information */
        ULONG ai_Pad1;
        ULONG ai_Pad2;
        BPTR ai_ProgDir;                /* base directory for application */
        BPTR ai_ConfigDir;              /* configuration file directory */
        struct List ai_PrefList;        /* list of preference files */
        STRPTR ai_BaseName;             /* pointer to application base name */
        STRPTR ai_ProgName;             /* pointer to application program name */
        STRPTR ai_AppName;              /* pointer to application name */
        STRPTR ai_AppVersion;           /* pointer to version string */
        STRPTR ai_AppCopyright;         /* pointer to copyright notice */
        STRPTR ai_AppAuthor;            /* pointer to author */
        STRPTR ai_AppMsgTitle;          /* pointer to title for messages */

        /* project information */
        struct Project ai_Project;      /* embedded Project structure */

        /* application information */
        APTR ai_UserData;               /* UserData */

        /* READ ONLY Intuition-specific information */
        STRPTR ai_ScreenName;           /* pointer to public screen name */
        struct Screen *ai_Screen;       /* Active screen */
        struct TextFont *ai_Font;       /* Font for screen */
        struct Window *ai_Window;       /* Active window */
        struct Gadget *ai_Gadget;       /* Active gadget */
        struct MHObject *ai_CurObj;     /* Active object (gadget) */
        struct DrawInfo *ai_DI;         /* Intuition DrawInfo */
        VOID *ai_VI;                    /* GadTools VisualInfo */
        WORD ai_MouseX;                 /* Position at last IDCMP message */
        WORD ai_MouseY;                 /* Position at last IDCMP message */

        /* AppShell-maintained fields */
        /* the remainder of the fields are private to the AppShell */
    };
```

The AppInfo structure contains the following components:

The following fields can be accessed by message handlers or functions and are used for control purposes.:

ai_TextRtn - Used by message handler initialization routines to return a string to indicate an error. Used by functions to return text-based information when there isn't an error.

ai_Pri_Ret - Used by functions to return an error level. Use the levels provided in *<dos/dos.h>*

ai_Sec_Ret - Used by functions to return an error information. Set to the index of the error in the error text array.

ai_Done - Set to TRUE by the application's Quit function, to indicate to the event processor that the user wants to shut the application down.

The following fields provide information on the startup environment of the application:

ai_Startup - This field contains information on the startup of the application. If APSHF_START_WB bit is set, then the application was started from the Workbench, otherwise it was started from the Shell. If the APSHF_START_CLONE field is set, then the application is a cloned application, started by the Tool handler.

ai_Options - If the application was started from the Shell, then this field contains the parsed command line (according to ReadArgs). If no template was specified, then AppShell defaults to FILES/m.

ai_NumOpts - Tells how option buckets are valid in ai_Options.

ai_ArgsPtr - ReadArgs pointer.

ai_FileArray - If the first option in the command template is a multi-arg, then this points at the list of arguments. For example, if the template was "FILES/m, ALL/s", then ai_FileArray would be a list of the file names that were passed.

ai_ProgDO - This is the tool icon for the application. This is valid whether called from Shell or Workbench.

The following fields are for obtaining information on the application itself:

ai_ProgDir - Lock on the directory that the application resides in.

ai_ConfigDir - Lock on the directory that contains the preference files for the application.

ai_PrefList - List of preference records that the AppShell has loaded.

ai_BaseName - Base name of the application. This name is used to prefix preference files, base name of public message ports and other public names for the application.

ai_ProgName - Application program name.

ai_AppName - Application name.

ai_AppVersion - Application version.

ai_AppCopyright - Application copyright notice.

ai_AppAuthor - Application author.

ai_AppMsgTitle - Title to display in the message requester.

The following field is for tracking project related information:

ai_Project - Pointer to the project structure. Each project is represented by a project node. Projects are arguments that are passed from Workbench at startup or FILE/m from Shell. The application can also have AppIcon, AppMenu and AppWindow arguments added to this master project list.

The following field is for application specific data:

ai_UserData - Application specific user data.

The following fields are maintained by the IDCMP message handler and are *read only*. Some of the more timely items are only fresh when the function was triggered by an IDCMP message.

ai_ScreenName - If the application is residing on a public screen, then this is the name of that screen.

ai_Screen - Screen for all the applications' windows.

ai_Font - Font used for text items inside the window.

ai_Window - Pointer to the window that was active when the function was invoked.

ai_Gadget - Pointer to the gadget that was active when the function was invoked.

ai_CurObj - Pointer to the Object that was active when the function was invoked gadget).

ai_DI - DrawInfo for the applications screen (used by Intuition).

ai_VI - VisualInfo for the applications screen (used by GadTools).

ai_MouseX
ai_MouseY - Coordinates of the mouse when the function was invoked.

The programmer should lock the structure before modifying any of the fields, and unlock it when done. This is done using the LockAppInfo() and UnlockAppInfo() functions.

Project List

The AppShell provides the capability to maintain multiple projects. These projects could be maintained as single threads or multi-threads. A single threaded application, for example, could be a print spooler that steps through the list of projects. A multi-threaded application, for example, could be a text editor that uses a different process for each project.

Many times a project consists of multiple components. For example, an animation consists of several frames. Another example would be a multimedia application which could have a list of pictures, another list of sounds, etc.

To maintain the list of projects and its list of components, the Project structure is used.

```
struct Project
{
      struct List p_ProjList;         /* Project list */
      struct ProjNode *p_CurProj;     /* Current project */
      LONG p_NumProjs;                /* Number of projects */
      LONG p_MaxID;                   /* Next available ID */
      LONG p_State;                   /* Listview state */
      LONG p_TopView;                 /* Listview top line */
      LONG p_CurLine;                 /* Listview current line */
      ULONG p_Flags;                  /* Project flags */
      APTR p_UserData;                /* User data extension */
      APTR p_SysData;                 /* System data extension */
};
```

The Project structure consists of the following fields:

p_ProjList - A list of ProjNode's that contain information on each project.

p_CurProj - Current project (usually only applicable for single-threaded applications).

p_NumProjs - Number of projects in the list.

p_MaxID - Next available ID. These ID's are assigned to projects so that their order can be manipulated.

p_State
p_TopView
p_CurLine - Used when displaying the projects in a Scrolling List gadget.

p_Flags - Used to indicate information about the project list.

APSHF_PROJVIEW - Indicates that the project list is being displayed, currently, by the Project List data entry requester.

p_UserData - Maintain project oriented data.

p_SysData - Reserved for system use.

The AppShell uses the ProjNode structure to maintain information on the projects that the application is working on. Here is what the ProjNode structure looks like:

```
struct ProjNode
{
    struct Node pn_Node;              /* embedded Exec node */

    /* AppShell information.  Read only for application */
    struct DateStamp pn_Added;        /* date stamp when added to list */
    BPTR pn_ProjDir;                  /* lock on project directory */
    STRPTR pn_ProjPath;               /* pointer to the projects' complete name */
    STRPTR pn_ProjName;               /* pointer to the projects' name */
    STRPTR pn_ProjComment;            /* pointer to the projects' comment */
    struct DiskObject *pn_DObj;       /* pointer to the projects' icon */
    LONG pn_ID;                       /* user selected order */
    APTR pn_SysData;                  /* System data extension */

    /* Application information */
    ULONG pn_Status;                  /* status of project */
    ULONG pn_ProjID;                  /* project ID */
    UBYTE pn_Name[32];                /* project name */
    APTR pn_UserData;                 /* UserData for project */
    BOOL pn_Changed;                  /* has project been modified? */
};
```

The following fields are maintained by the AppShell and are available as read only to the application:

pn_Node - Embedded Exec node.

pn_Added - Date stamp of when the project was added to the project list. Currently not implemented.

pn_ProjDir - Lock on the directory that the project is located in. Maintained by the AppShell, NOT to be unlocked by the application.

pn_ProjPath - Pointer to the complete name, including path, of the project.

pn_ProjName - Pointer to the name of the project.

pn_ProjComment - Pointer to the file comment for the project.

pn_DObj - Pointer to the icon (DiskObject) for this project.

pn_ID - ID used to sort the project list by user selected order.

pn_SysData - Reserved for AppShell use only.

The following fields belong to the application:

pn_Status - Status of the project.

pn_ProjId - Project ID.

pn_Name - Name of the project.

pn_UserData - Application specific data.

pn_Changed - Has the project been edited since last save?

Library Handling

The AppShell provides ways to open and close a list of shared system libraries. Libraries can be marked as optional or required. AppShell will shut down if the library is required and can't be opened.

An example of the data and tags required for maintaining a number of libraries would be:

```
/* required libraries */
extern struct Library *SysBase;     /* required for pragmas */
struct Library *GadToolsBase;
struct Library *GfxBase;
struct Library *IconBase;
struct Library *IntuitionBase;
struct Library *UtilityBase;
struct Library *MySpecialBase;

/* This tag array is used to open and close the shared system libraries
 * needed by our application. */
struct TagItem Our_Libs[] =
{
    {APSH_LibVersion,       36L},           /* Minimum library version */
    {APSH_LibStatus,        APSH_REQUIRED}, /* All libraries are required */
    {APSH_GadTools,         (ULONG) &GadToolsBase},
    {APSH_Gfx,              (ULONG) &GfxBase},
    {APSH_Icon,             (ULONG) &IconBase},
    {APSH_Intuition,        (ULONG) &IntuitionBase},
    {APSH_Utility,          (ULONG) &UtilityBase},
    {APSH_LibName,          (ULONG)"myspecial.library"},
    {APSH_LibBase,          (ULONG) &MySpecialBase},
    {TAG_DONE,}
};

/* Tell about our application */
struct TagItem Our_App[] =
{
    /* Trigger the library opening module */
    {APSH_OpenLibraries,    (ULONG)Our_Libs},

    /* ... other application tags ... */

    {TAG_DONE,}
};

struct Library *AppShellBase;
extern struct WBStartup *WBenchMsg;

/* Main processing loop */
VOID main (int argc, char **argv)
{
    /* open the AppShell library */
    if (AppShellBase = OpenLibrary ("appshell.library", 36))
    {
        /* Main AppShell entry point */
        HandleApp (argc, argv, WBenchMsg, Our_App);

        /* close the AppShell library */
        CloseLibrary (AppShellBase);
    }
}
```

The library module understands the following tags:

APSH_LibVersion - Used to indicate the minimum library version required. This version level is in effect until another APSH_LibVersion is specified.

APSH_LibStatus - Indicate whether the application requires the library or not. APSH_REQUIRED or APSH_OPTIONAL are the only valid values. This status is in effect until another APSH_LibStatus is specified.

APSH_LibNameTag - A tag is used to specify the library to open. Following is the list of available library tags.

APSH_ARexxSys
APSH_ARexxSup
APSH_ASL
APSH_Commodities
APSH_DiskFont
APSH_DOS
APSH_GadTools
APSH_Gfx
APSH_Icon
APSH_Intuition
APSH_Layers
APSH_IFF
APSH_Translate
APSH_Utility
APSH_Workbench
APSH_AppObjects
APSH_Hyper
APSH_Prefs

APSH_LibName - Use this tag to specify a library name that isn't already defined by a name tag. For example,

```
{APSH_LibName, (ULONG)"myspecial.library"},
```

APSH_LibBase - A pointer to the library base variable name. This tag is what triggers a library open, so must be placed after any of the Lib tags mentioned above.

Startup Processing

The AppShell obtains the following application information once the shared system libraries have been opened and before the message handlers have been initialized:

Home directory
Preference file directory
Base name
Program name
Tool icon
Arguments

This information is obtained whether the application was called from the Shell or from Workbench. If the application was invoked from Workbench, then icon tool type parsing applies. Otherwise, the standard Shell ReadArgs is used. If multiple projects were passed, they are appended to the project list.

The AppShell provides a standard ReadArgs template for applications that doen't specify their own. This template is as follows:

Files/M,PubScreen/K,PortName/K,Startup/K,NOGUI/S

Where:

Files/M - Files to use as projects. Supports wildcard expansion.

PubScreen/K - Name of the existing public screen to open on.

PortName/K - Name to assign to the ARexx port of the application.

Startup/K - Name of an ARexx script to run at startup (before entering the main loop of the application).

NOGUI/S - Disable the graphical user interface of the application. Usefull for running scripts that are to be run in the background.

Note that these keywords are reserved words, and their presence will be detected by the AppShell (regardless of order of appearance in the template).

Here is a code fragment showing how the startup arguments are passed to the AppShell. Note that all Workbench arguments are automatically added to the master project list.

```
/* Shell template */
#define TEMPLATE "Files/M,Name/K"
#define OPT_FILES   0
#define OPT_NAME    1
#define OPT_COUNT   2

/* Sample tags */
struct TagItem Our_App[] =
{
    {APSH_Template, (ULONG) TEMPLATE},
    {APSH_NumOpts, (ULONG) OPT_COUNT},

    /* ... other tags ... */

    {APSH_AppInit, CInitID},
    {TAG_DONE,}
};

extern struct WBStartup *WBenchMsg;
struct Library *AppShellBase;

/* Main processing loop */
VOID main (int argc, char **argv)
{
    /* open the AppShell library */
    if (AppShellBase = OpenLibrary ("appshell.library", 36))
    {
        /* Main AppShell entry point */
        HandleApp (argc, argv, WBenchMsg, Our_App);

        /* close the AppShell library */
        CloseLibrary (AppShellBase);
    }
}

/* Sample initialization function */
VOID
CInitFunc (struct Hook *h, struct AppInfo * ai, struct AppFunction *af)
{
    /* See if there is an array of files */
    if (ai->ai_Options[OPT_FILES])
    {
        /* Do stuff with the file array (it has already been expanded
         * and added to the project list). */
    }

    /* See if the name was specified */
    if (ai->ai_Options[OPT_NAME])
    {
        /* Do stuff with name */
    }
}
```

The following tags are used to specify startup information.

APSH_Template - Used to specify the ReadArgs template to use when the application is started from the Shell.

APSH_NumOpts - The number of options specified in the template (APSH_Template).

Function Table Entries

All functions that require a user interface are placed in an array called the Function Table. The function table is built from entries made up by the Funcs structure.

The Funcs structure contains the variables required for a function table entry. Here is what the Func structure contains:

```
/* Function table entry */
struct Funcs
{
    UBYTE *fe_Name;              /* Name of function */
    VOID (*fe_Func)(struct Hook *,struct AppInfo *,struct AppFunction *);
    ULONG fe_ID;                 /* ID of function */
    STRPTR fe_Template;          /* Command template */
    ULONG fe_NumOpts;            /* Number of options */
    ULONG fe_Flags;              /* Status of function */
    ULONG fe_HelpID;             /* Index into the the text catalogue for help */
    STRPTR fe_Params;            /* Optional parameters for function */
    ULONG *fe_GroupID;           /* ~0 terminated array of group ID's */
    LONG *fe_Options;            /* ReadOnly! ReadArgs */
};
```

The Funcs structure contains the following components:

fe_Name - A pointer to the short name of the function. This is the name that would be used in bindings or scripts.

fe_Func - Pointer to actual function invoked by the event processor.

fe_ID - Unique number assigned to the function. This is the number used in bindings; such as menu item, gadgets or hot keys.

fe_Template - This is the ReadArgs template that this function uses for option parsing.

fe_NumOpts - Number of options described in fe_Template.

fe_Flags - Status of the function entry. The following flags are available to the application.

APSHF_PRIVATE - Indicate that function can not be called by the user. Your initialization and shutdown routines should have this flag set.

APSHF_LOCKON - Indicate that the function can not be disabled. Usefull for protecting certain functions from being modified by the command "DISABLE #?".

fe_HelpID - Index into text catalogue for single line help on function. Also used for setting the context for hyper.library (the HyperText system).

fe_Params - When present, these are the actual parameters that would be passed to the function whenever it is invoked. This field is generally initialized by the ALIAS function.

fe_GroupID - Pointer to a non-zero terminated array of group ID's that this function entry belongs in. Currently not implemented.

fe_Options - This read-only field is set up by the AppShell function dispatcher by a call to ReadArgs using the template supplied in fe_Template.

When a function is invoked, one of the arguments it receives is a pointer to an AppFunction structure. The AppFunction structure is made up of the following fields.

```
/* AI_FUNCTION */
struct AppFunction
{
    ULONG MethodID;
    STRPTR af_CmdLine;              /* Untouched command line */
    struct TagItem *af_Attrs;       /* Attributes */
    LONG af_Kind;                   /* ID of the active message handler */
    struct Message *af_Msg;         /* Pointer to the active message */
    struct Funcs *af_FE;            /* Current function entry */
};
```

The following is a description of the fields within the AppFunction structure:

MethodID - This field is used to indicate the format of the remaining fields of the message. Currently the AppShell only passes AI_FUNCTION messages.

af_CmdLine - The command string that was used to invoke the function.

af_Attrs - An optional list of TagItems, containing additional parameters.

af_Kind - This field indicates the user interface that invoked the message. The following kinds are defined in *<libraries/appshell.h>*

```
APSH_AREXX_IDARexx
APSH_DOS_ID  Command Shell
APSH_IDCMP_IDIntuition
APSH_SIPC_ID Simple Interprocess Communications
APSH_TOOL_ID Asynchronous Process
APSH_WB_ID   Workbench (AppIcon/AppMenu/AppWindow)
```

af_Msg - A pointer to the message that invoked the function. Not to be modified in any way.

af_FE - A pointer to the FuncEntry for the function.

Here is an example of how an application would set up the function table:

```
/* These are the function prototypes for all the application implemented  functions used in  * this
```

```
example. */
VOID NewFunc  (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID OpenFunc (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID BModListFunc (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID AModListFunc (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID QuitFunc (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID OkayFunc (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID PreviousFunc (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID NextFunc (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID RemoveFunc    (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID CancelFunc    (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID SelectFunc    (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID ListFunc (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID SwapFunc (struct Hook *,struct AppInfo *,struct AppFunction *);
VOID HotkeyFunc    (struct Hook *,struct AppInfo *,struct AppFunction *);

/* Each public function gets a numeric ID assigned to it.  */
#define DUMMYID APSH_USER_ID
#define OkayID   (DUMMYID + 1L)
#define MyPrevID(DUMMYID + 2L)
#define MyNextID(DUMMYID + 3L)
#define MyRemvID(DUMMYID + 4L)
#define CancelID(DUMMYID + 5L)
#define ListID   (DUMMYID + 6L)
#define SwapID   (DUMMYID + 7L)
#define BModListID   (DUMMYID + 8L)
#define AModListID   (DUMMYID + 9L)
#define Macro1ID(DUMMYID + 10L)
#define HotkeyID(DUMMYID + 11L)
#define LAST_ID  (DUMMYID + 12L)

/* The AppShell will convert this array into function table entries and will add them to the function
 * table list.  Using the APSHF_PRIVATE flag, you are able to have functions that can't be triggered
 * by the user. */
    struct Funcs FTable[] =
    {
        {"New",      NewFunc,    NewID,},
        {"Open",     OpenFunc,   OpenID,},
        {"Quit",     QuitFunc,   QuitID,},
        {"Okay",     OkayFunc,   OkayID,},
        {"Previous",PreviousFunc,MyPrevID,},
        {"Next",     NextFunc,   MyNextID,},
        {"Remove",   RemoveFunc, MyRemvID,},
        {"Cancel",   CancelFunc, CancelID,},
        {"Select",   SelectFunc, SelectID,},
        {"Swap",     SwapFunc,   SwapID,},
        {"BMod",     BModListFunc,BModListID,},
        {"AMod",     AModListFunc,AModListID,},
        {"Key",      HotkeyFunc, HotkeyID,},

        /* This function can not be accessed by the user */
        {"List",     ListFunc,   ListID, NULL, NULL, APSHF_PRIVATE},

        /* This function calls an ARexx macro */
        {"Macro1",   NO_FUNCTION, Macro1ID,},

        /* Marks the end of the array */
        {NULL,       NO_FUNCTION,}
    };

    /* Sample tags */
    struct TagItem Our_App[] =
    {
        /* This tag indicates that the functions all use the
         * recommended Hook interface. THIS TAG IS REQUIRED! */
        {APSH_HookClass, TRUE},

        /* This tag is used to specify the function table */
        {APSH_FuncTable, (ULONG)FTable},

        /* ... other tags ... */

        {TAG_DONE,}
    };

    extern struct WBStartup *WBenchMsg;
    struct Library *AppShellBase;

    /* Main processing loop */
    VOID main (int argc, char **argv)
    {
        /* open the AppShell library */
        if (AppShellBase = OpenLibrary ("appshell.library", 36))
        {
            /* Main AppShell entry point */
            HandleApp (argc, argv, WBenchMsg, Our_App);

            /* close the AppShell library */
            CloseLibrary (AppShellBase);
        }
    }
```

The new tags used in this example are:

APSH_FuncTable - Pointer to an array of Funcs structures.

Note that the last NO_FUNCTION is required to mark the end of the function table.

The application function ID's must start at the predefined (defined in *<libraries/appshell.h>*) value APSH_USER_ID. Once the function table has been defined, actual application functions need to be established. In order to separate the user interface from the functionality of the application, a common function interface needs to be defined. Necessary information has to be passed to the function regardless of what event triggered the function. Each standard AppShell and Application function has a common argument interface.

```
/* sample function declaration */
StubFunc (h, ai, af)

struct Hook *h;             /* Standard hook interface */
struct AppInfo * ai;        /* Standard application variables */
struct AppFunction * af;    /* Message containing arguments */
```

The parameters for the common argument interface are:

h - A pointer to the Hook structure that was used to invoke the function. This structure is to be ignored by the function.

ai - A pointer to the AppInfo structure that contains all the global data required for this application.

af - A pointer to the AppFunction structure that contains the arguments for the function.

For example, a function with a function entry of;

```
{"Remove", RemoveFunc, RemoveID, "NAME/K,ALL/S", 2L,},
```

could be written in the following manner:

```
RemoveFunc (struct Hook *h, struct AppInfo *ai, struct AppFunction *af)
{
    struct TagItem *attrs = af->af_Attrs;
    STRPTR name = NULL;
    BOOL all = FALSE;
    struct Funcs *f;

    /* See if the function arguments have been parsed */
    if (f = af->af_FE)
    {
        /* Was a name given? */
        name = (STRPTR) f->fe_Options[0];

        /* Was the all switch set? */
        all = (BOOL) f->fe_Options[1];
    }
    else
    {
        /* See if a name was passed */
        name = (STRPTR) GetTagData (APSH_NameTag, NULL, attrs);
    }

    if (all)
    {
        /* They want us to remove all objects. */
    }
    else if (name)
    {
        /* They just want a particular object removed. */
    }
}
```

Error and Text Handling

The application should define the possible text messages that could be used and build an array of messages. Then whenever a message is needed, it will be referenced by an index into the array.

If an error is encountered, the index value for the message is placed in ai_Sec_Ret, the error level is placed in ai_Pri_Ret, and the text is placed in ai_TextRtn. The AppShell will then make sure that user is notified of the error in an appropriate way. For example, if the command that encountered the error was triggered directly by the IDCMP message handler, then an EasyRequester will be displayed.

Use the values that are assigned in *<dos/dos.h>* for the severity level.

```
RETURN_OK 0       /* No problems, success */
RETURN_WARN   5   /* A warning only */
RETURN_ERROR 10   /* Something wrong */
RETURN_FAIL  20   /* Complete or severe failure*/
```

Functions that return text, via ARexx, must set both ai_Pri_Ret and ai_Sec_Ret to zero and point ai_TextRtn to the text return value.

Following is a code fragment that shows how to set up the default text table. Note that the table does not contain any embedded NULLs.

```
/* application error table */
STRPTR Def_Text[] =
{
  "<unassigned>",         /* padding */
  "Illegal transformation on %s",
  NULL        /* NULL termination is required */
};

/* easy to remember define */
#define ERROR_TRANSFORM 1

/* sample application tags */
struct TagItem Our_App[] =
{
    {APSH_FuncTable, FTable},
    {APSH_DefText,   Def_Text},
    ...
    {TAG_DONE,}
};

extern struct WBStartup *WBenchMsg;
struct Library *AppShellBase;

/* Main processing loop */
VOID main (int argc, char **argv)
{
    /* open the AppShell library */
    if (AppShellBase = OpenLibrary ("appshell.library", 36))
    {
        /* Main AppShell entry point */
        HandleApp (argc, argv, WBenchMsg, Our_App);

        /* close the AppShell library */
        CloseLibrary (AppShellBase);
    }
}

/* sample function returning an error */
VOID
StubFunc (struct Hook *h, struct AppInfo *ai, struct AppFunction *af)
{
    STRPTR name;

    ...

    /* sample error return */
    ai->ai_Pri_Ret = RETURN_WARN;
    ai->ai_Sec_Ret = ERROR_TRANSFORM;
    ai->ai_TextRtn = PrepText (ai, APSH_USER_ID, ERROR_TRANSFORM, name);
}
```

The new tags used in this example are:

APSH_DefText - Pointer to the default text table for this application.

In the above example, when StubFunc returns, the AppShell will display the text in ai_TextRtn to the user. If the function was triggered from a graphical user interface, then the message is displayed with an EasyRequest. If the function was triggered from ARexx, then the message is returned via a stem variable, <basename>.LASTERROR.

Message Handler Initialization

A message handler is a module that has been designed to manage messages, and therefore events, for a particular library or device. For example, the Intuition message handler manages IntuiMessages for objects, such as gadgets and menus attached to windows.

The AppShell maintains a list of message handlers. It starts by first calling the initialization routine of each message handler in the list. These message handlers can be marked as optional or required. The AppShell will shut down if a required message handler could not be initialized.

The following is a code fragment that outlines what is necessary to implement an optional ARexx message handler in your application:

```
/* ARexx user interface environment specification array */
struct TagItem Handle_AREXX[] =
{
    {APSH_Extens,    (ULONG)"proj"},
    {APSH_Rating,    APSH_OPTIONAL},
    {TAG_DONE,}
};

/* Tell about our application */
struct TagItem Our_App[] =
{
    {APSH_FuncTable,          (ULONG) Ftable},
    {APSH_DefText,            (ULONG) Def_Text},
    {APSH_OpenLibraries,      (ULONG) Our_Libs},
    {APSH_AddARexx_UI,        (ULONG) Handle_AREXX},
    {TAG_DONE,}
};

struct Library *AppShellBase;
extern struct WBStartup *WBenchMsg;

/* Main processing loop */
VOID main (int argc, char **argv)
{
    /* open the AppShell library */
    if (AppShellBase = OpenLibrary ("appshell.library", 36))
    {
        /* Main AppShell entry point */
        HandleApp (argc, argv, WBenchMsg, Our_App);

        /* close the AppShell library */
        CloseLibrary (AppShellBase);
    }
}
```

The new tags used in this example are:

APSH_AddARexx_UI - Add an ARexx user interface to our application. Data points to the ARexx user interface environment specification.

APSH_Extens - ARexx macro file name extension (see the ARexx section for more information).

The message handler initialization module understands the following tags:

APSH_AddHandler - Allow the specification of a custom message handler. Data points to a user interface environment specification.

APSH_AddArexx_UI - Specify the ARexx user interface.

APSH_AddClone_UI - Specify the clone application ability.

APSH_AddCmdShell_UI - Specify the Command Shell user interface.

APSH_AddIntui_UI - Specify the graphical user interface.

APSH_AddSIPC_UI - Specify the simple interprocess communication user interface.

APSH_AddTool_UI - Specify the tool user interface.

APSH_AddWB_UI - Specify the Workbench user interface.

These are tags that can be used in the user interface environment specification array:

APSH_Setup - Only used with the APSH_AddHandler tag to specify the function used to initialize the custom message handler.

APSH_Status - Set flags that control how the message handler is used.

APSHP_INACTIVE - When this flag is set, the handler won't be action until it receives a MH_OPEN message. If this flag isn't set, then the handler becomes immediately available.

APSHP_SINGLE - When this flag is set, then only one occurance of the message handler main object is allowed. When the flag isn't set, then multiple occurences are allowed. This only applies to the SIPC and ARexx interfaces.

APSH_Rating - This tag is used to specify whether the message handler is optional or required. The valid settings are APSH_REQUIRED and APSH_OPTIONAL.

APSH_Port - Specify the message port name for the message handler. This should not be used normally. You should allow the AppShell to name the message port on its own.

Standard Message Handlers

The following is an overview of each of the message handlers that are provided in the AppShell shared library. Information is given on:

- Short name of the message handler.
- List of tags that the message handler can translate.
- Standard Functions that the message handler implements.
- Extended low-level functions that the message handler implements.
- The preference files that the message handler gets user settings from.
- A short example on how an application can utilize the message handler.

Even though it isn't a message handler, information on the AppShell is also given due to the number of standard functions that it provides.

AppShell

The AppShell adds quite a number of standard functions to the function table. It also provides a number of function entry points.

Message Handler Name

<none>

Tags

APSH_NumArgs - This tag is used to specify the number of Shell arguments passed. This tag is used by the Clone or Tool feature and should not be used otherwise.
APSH_ArgList - This tag is used to specify the Shell arguments passed. This tag is used by the Clone or Tool feature and should not be used otherwise.
APSH_WBStartup - This tag is used to specify the Workbench arguments passed. This tag is used by the Clone or Tool feature and should not be used otherwise.
APSH_ControlPort - This tag is used to specify the SIPC control port for the application. This tag is used by the Clone or Tool feature and should not be used otherwise.
APSH_Name
APSH_Version
APSH_Copyright
APSH_Author - Used to specify the information about the application. To be - shown in the Standard About requester along with the image for ai_ProgDO.
APSH_AppMsgTitle - Used to specify the title to display in the error and message EasyRequest.
APSH_DefText - Specify the default text table for the application.
APSH_AppInit - Function ID to dispatch after the message handlers have been initialized, and before entering the event processing stage.

APSH_AppExit - Function ID to dispatch after ai_Done has been set to TRUE, and before running shut down on the message handlers.

APSH_SIG_C - Function ID to dispatch when the corresponding signal is sent to the application. Defaults to QuitID.

APSH_SIG_D

APSH_SIG_E

APSH_SIG_F - Function ID to dispatch when the corresponding signal is sent to the application.

APSH_BaseName - Base name assigned to the application. The public screen name, ARexx message port and other public names are derived from this.

APSH_Template - ReadArgs template to use when the application is started from the Shell.

APSH_NumOpts - Number of options specified in APSH_Template.

APSH_UserData - Pointer to preallocated user data for the application. ai_UserData will be set to this value.

APSH_UserDataSize - If this tag is specified, then the AppShell will allocate this much memory off of the ai_UserData field at initialization time, and free it at shutdown.

Standard Functions

Alias
NAME,COMMAND/F
Link a new command name to an old command name.

Disable
/M
Disable a function or a set of functions. If there is a GUI item, then it will disable that item also, IE if the function is bound to a menu item, then it disables that menu item. Currently only disables functions and gadgets. Supports multiple names and pattern matching.

Enable
/M
Enable a function or a set of functions. Currently only enables functions and gadgets. Supports multiple names and pattern matching.

Fault
/N
Return the text string assigned to an error number.

Stub
,
Function which does nothing.

Version
APPSHELL/S
Display the version string of the application. When the APPSHELL switch is set, then display the version of the AppShell.

ARexx Message Handler

The ARexx message handler provides a standard scripting language, as well as string oriented interprocess communications. Currently the ARexx message handler only implements commands.

Message Handler Name

AREXX

Tags

APSH_Extens - ARexx macro file name extension. Defaults to .REXX.
APSH_ARexxError - Function ID to dispatch when there is an ARexx command error.
APSH_ARexxOK - Function ID to dispatch when an ARexx command succeeds.
APSH_Port - Base name for the applications public ARexx port. Defaults to the application's base name.
APSH_Status - Recognizes:
APSHP_INACTIVE - When set, leaves the ARexx message handler inactive. Causes the message handler to reply to each incoming message with the return value set to RETURN_FAIL. When not set, causes ARexx to become immediately available.
APSHP_SINGLE - When set, uses the base port name as is. When not set, will append .# to the base port name, where # is incremented until it finds a unused name.

Extended Low-Level Message Handler Functions

AH_SENDCMD - Sends the passed command (APSH_CmdString) to ARexx.

Standard Functions

RX
COMMAND/f
Execute an ARexx command. Only to be used when there is a name conflict between an ARexx command name and a function short name.

Why
,
Return more information on the last error. The information is placed in the ai_TextRtn field, and could be text or a number sprintf()ed into text. In an ARexx script, the primary return value should indicate failure level. This command allows us to get at the real reason a command may have failed. This function is currently not implemented.

Preferences

<none>

Example

```
/* ARexx user interface environment specification array */
struct TagItem Handle_AREXX[] =
{
    {APSH_Extens,    (ULONG)"proj"},
    {APSH_Rating,    APSH_OPTIONAL},
    {TAG_DONE,}
};

/* Tell about our application */
struct TagItem Our_App[] =
{
    {APSH_AddARexx_UI,       (ULONG)Handle_AREXX},

    /* ... other tags go here... */

    {TAG_DONE,}
};

struct Library *AppShellBase;
extern struct WBStartup *WBenchMsg;

/* Main processing loop */
VOID main (int argc, char **argv)
{
    /* open the AppShell library */
    if (AppShellBase = OpenLibrary ("appshell.library", 36))
    {
        /* Main AppShell entry point */
        HandleApp (argc, argv, WBenchMsg, Our_App);

        /* close the AppShell library */
        CloseLibrary (AppShellBase);
    }
}
```

Command Shell Message Handler

The Command Shell Message Handler provides a command shell for the more advanced user. It allows the user to directly dispatch any of the functions in the function table. It also gives direct access to the power of ARexx without always having to specify the ADDRESS or macro file name extension.

Message Handler Name

DOS

Tags

APSH_CloseMsg - Text ID of the message to display in the Shell before attempting to close it. Defaults to "Waiting for macro return".

APSH_CMDWindow - Text ID of the initial Shell window specification. Defaults to "CON:0/150/600/50/Command Shell/CLOSE/AUTO".

APSH_Prompt - Text ID of the initial Shell prompt. Defaults to "Cmd> ".

APSH_CMDTitle - Used to specify the text ID of title for the Command Shell. Defaults to <basename> Command Shell.

APSH_CMDParent - Used to specify the name of the parent window of the Command Shell. Should be the main project window of the application. Using this tag helps the AppShell keep the Command Shell near the application when the user is using a virtual screen. Not fully implemented yet.

APSH_Status - Recognizes:
APSHP_INACTIVE - If set, keep the Command Shell window closed until specifically requested to open. If not set, then open the Command Shell window at initialization time.

Extended Low-Level Message Handler Functions

<none>

Standard Functions

CmdShell
OPEN/s,CLOSE/s,TITLE/k,SNAPSHOT/s,ACTIVATE/s,FRONT/s,BACK/s
CmdShell allows the user to manipulate the Command Shell.

OPEN
Opens the Command Shell window, respecting the current snapshot.

CLOSE
Closes the Command Shell window.

TITLE
Set the Command Shell's window title bar.

SNAPSHOT
Save the current rectangle of the window to a file.

ACTIVATE
Activate the Command Shell window.

FRONT
Bring the Command Shell window to the front of other windows.

BACK
Send the Command Shell window to the back of the other windows.

Preferences

commandshell.win - Window size and placement preferences.

Example

```
/* Command Shell user interface environment specification array */
struct TagItem Handle_DOS[] =
{
    {APSH_Status,   APSHP_INACTIVE},
    {APSH_Rating,   APSH_OPTIONAL},
    {TAG_DONE,}
};

/* This is the main tag list used to describe our application's multiple
```

```
 * user interfaces. */
struct TagItem Our_App[] =
{
    {APSH_AddCmdShell_UI, (ULONG)Handle_DOS},

    /* ... other tags go here ... */

    {TAG_DONE, }
};

struct Library *AppShellBase;
extern struct WBStartup *WBenchMsg;

/* Main processing loop */
VOID main (int argc, char **argv)
{
    /* open the AppShell library */
    if (AppShellBase = OpenLibrary ("appshell.library", 36))
    {
        /* Main AppShell entry point */
        HandleApp (argc, argv, WBenchMsg, Our_App);

        /* close the AppShell library */
        CloseLibrary (AppShellBase);
    }
}
```

IDCMP Message Handler

The IDCMP message handler provides the application with a Graphical User Interface (GUI) through the use of Intuition, GadTools and AppObjects.

Maintains such things as:

- Public or private screens - Manages the DrawInfo and VisualInfo for the application screen. Also tracks the public screen information whenever appropriate.
- Multiple windows - Manages an unlimited number of application windows.
- Translating Workbench preferences - Translate user-interface object description, based on user preferences, into GadTools and Intuition objects. For instance, it will scale the objects according to the font; or remap pens according to the color palette. Will also load the normal and wait pointers.
- Snapshotting window preferences - Manages saving/restoring user preferences for individual windows. Includes such things as placement, and size.
- GadTool gadgets - Converts object descriptions into GadTools structures and tags.
- boopsi objects - Supports boopsi objects.
- Images, borders and text - Object management of images, text and common border types.
- define GUI items such as windows, buttons, sliders and scrolling lists, the application uses an array of Object structures. Here is what the Object structure looks like (defined in *<libraries/appshell.h>*):

```
struct Object
{
    struct Object *o_NextObject;      /* next object in array */
    WORD o_Group;                     /* Object group */
    WORD o_Priority;                  /* Inclusion priority of object */
    ULONG o_Type;                     /* type */
    ULONG o_ObjectID;                 /* ID */
    ULONG o_Flags;                    /* see defines below */
    UWORD o_Key;                      /* hotkey */
    STRPTR o_Name;                    /* name */
    ULONG o_LabelID;                  /* label index into text catalogue */
    struct IBox o_Outer;              /* size w/label */
    struct TagItem *o_Tags;           /* tags for object */
    APTR o_UserData;                  /* user data for object */
};
```

The Object structure contains the following components:

o_NextObject - Pointer to the next object in the array.
o_Group - The layout group that the object belongs in.
o_Priority - The priority assigned to the object or group. Zero indicates high priority.
o_Type - Indicates type of object. Valid object types are shown in the next section.
o_ObjectID - Function ID of the command to call when an event occurs for this object.
o_Flags - Flags which define the behavior of the object.
APSH_OBJF_CLOSEWINDOW - Indicates that this object will cause the window to close.
APSH_OBJF_ACTIVATE - Indicate that this is the default object to activate when the window becomes active.
o_Key - Keystroke binding for this object. Whenever the key is pressed, the corresponding function for the gadget is executed. On the downpress of the key, then EG_DOWNPRESS is executed, repeat calls EG_HOLD, and release calls EG_RELEASE. If the label for the object contains an '_' sign in it, then the key following the sign will become the keystroke binding for the object. This for keystroke internationalization also.
o_Name - All objects are placed into a list for the window that they belong in. o_Name is the name that is used for the list entry (node) for this object. This name should be unique to this window.
o_LabelID - A numeric ID assigned to the visual text label for this object.
o_Outer - The IBox which describes the placement and size of an entire object, including the visual label, if pertinent. A negative number for any value indicates that that value is relative. Currently the GadTools objects don't support relativity.
Left - When positive indicates LEFT, when negative indicates GRELRIGHT.
Top - When positive indicates TOP, when negative indicates GRELBOTTOM.
Width - When positive indicates WIDTH, when negative indicates GRELWIDTH.
Height - When positive indicates HEIGHT, when negative indicates GRELHEIGHT.
o_Tags - Additional parameters for defining the object. Following are the generic tags used by the gadget-like objects.
APSH_GTFlags - GadTools NewGadget flags to use for this object.
APSH_ObjDown - Function ID to dispatch on the downpress of an Intuition gadget.
APSH_ObjHold - Function ID to dispatch when an Intuition gadget is being held down.
APSH_ObjRelease - Function ID to dispatch on the release of an Intuition gadget.
APSH_ObjDblClick - Function ID to dispatch when an Intuition gadget has been double-clicked.
APSH_ObjAbort - Function ID to dispatch when the right mouse button has been pressed while an Intuition gadget is active.
APSH_ObjAltHit - Function ID to dispatch when an Intuition gadget is selected while holding either ALT key.
APSH_ObjShiftHit - Function ID to dispatch when an Intuition gadget is selected while holding either SHIFT key.
APSH_ObjExtraRelease - Used to specify the function for the button added to OBJ_DirString or OBJ_DirNumeric.
APSH_ObjCreate - Function ID to dispatch after the object has been created.
APSH_ObjDelete - Function ID to dispatch before the object may be deleted.
APSH_ObjUpdate - Function ID to dispatch when an IDCMP_IDCMPUPDATE event occurs for

the gadget. The AppShell searches for the object that is associated with the GA_ID attribute when the event occurs.

APSH_ObjData - Used to provide the additional data for the object. Used to specify the image for OBJ_Display, OBJ_Select, OBJ_Dropbox, OBJ_GImage, and OBJ_Image.

APSH_ObjAltData - Used to specify the alternate image for an object.

o_UserData - Pointer available to the application for binding data to a particular object. Note that the gadget's UserData field is currently used by the AppShell, and application user data is assigned to the ObjectNode's UserData field.

The following are the valid object types (defined in *<libraries/appshell.h>*). They are broken into type; GadTool, boopsi, image and group.

The tags that each object understands are listed below the object type. The tags are prefixed with a character that indicates when the tag can be specified. An 'I' indicates that it can be set at initialization time. A 'S' indicates that it can be manipulated with the SetGadgetAttrs() or GT_SetGadgetAttrs(). A 'G' indicates that it can be obtained with GetAttr(). A 'N' indicates that the attribute is sent out to the APSH_ObjUpdate function.

The following objects are GadTool gadgets. They accept the GadTool tags defined for the particular gadget type. Attributes are set with the GT_SetGadgetAtts() call.

OBJ_Generic

OBJ_Checkbox - Check box for boolean values. o_ObjectID is called on downpress. Height is font height.
IS) GTCB_Checked

OBJ_Integer - String gadget for numeric entry. o_ObjectID is called on downpress. Height is font height plus six.
I) GTIN_MaxChars
IS) GTIN_Number

OBJ_Listview - Scrolling list gadget. o_ObjectID is called on downpress. Minimum width of fifty pixels, minimum height of thirty pixels.
I) Use the APSH_ShowSelected tag to specify the name of an OBJ_Integer or OBJ_String gadget to attach to the bottom of the list view.
I) GTLV_ReadOnly
IS) GTLV_Top
IS) GTLV_Labels
IS) GTLV_Selected

OBJ_MX - Mutual exclusion gadget. o_ObjectID is called on downpress.
I) GTMX_Spacing
IS) GTMX_Labels
IS) GTMX_Active

OBJ_Number - Numeric display box. Height is font height plus six.

I) GTNM_Border
IS) GTNM_Number

OBJ_Cycle - Cycle gadget. o_ObjectID is called on release. Height is font height plus six. Minimum height of sixteen.

IS) GTCY_Labels
IS) GTCY_Active

OBJ_Palette - Color palette gadget. o_ObjectID is called on downpress.

IS) GTPA_Color

OBJ_Slider - Slider gadget.

IS) GTSL_Min
IS) GTSL_Max
IS) GTSL_Level
IS) GTSL_MaxLevelLen
IS) GTSL_LevelFormat
IS) GTSL_LevelPlace
IS) GTSL_DispFunc

OBJ_String - String gadget for text entry. o_ObjectID is called on RETURN. Height is font height plus six.

I) GTST_MaxChars
IS) GTST_String

OBJ_Text - Text display box. Height is font height plus six.

I) GTTX_Border
IS) GTTX_Text
IS) GTTX_CopyText

The following gadgets are boopsi objects. Attributes are set with the Intuition SetGadgetAttrs() function. Attributes may also be read by using the GetAttrs() function.

OBJ_Button - Action gadget. o_ObjectID is called on release. Height is font height plus six. Label is centered within object.

I) Uses o_LabelID for the text to display.
SG) GA_TEXT

OBJ_Scroller - Scroll gadget. o_ObjectID is called on release, and for changes.

ISGN) GTSC_Top - Top line of the view.
ISGN) GTSC_Total - Total number of lines.
ISGN) GTSC_Visible - Number of lines in view.
ISGN) GTSC_Overlap - Amount of overlap when scrolling pages.

S) CGTA_Increment - Increment top by one.
S) CGTA_Decrement - Decrement top by one.

OBJ_Display - Freeform display box for text or images.
I) Uses o_LabelID for the text to display.
SG) GA_TEXT
I) Specify the APSH_ObjData tag in order to use an Intuition image.
SG) GA_LABELIMAGE

OBJ_Select - Freeform action gadget for text or images.
I) Uses o_LabelID for the text to display.
SG) GA_TEXT
I) Specify the APSH_ObjData tag to use an Intuition image instead.
SG) GA_LABELIMAGE
I) In order to use a boopsi image, use the APSH_GA_LabelImage tag to provide the name of an OBJ_boopsi object.
SG) GA_LABELIMAGE

OBJ_Dropbox - AppWindow icon drop box. Specify the APSH_ObjData to specify an image to display. Size is ninety pixels wide by fourty-four pixels high.
I) Use the APSH_ObjData tag to specify the initial image to display.
SG) GA_LABELIMAGE
I) Set the APSH_OBJF_DRAGGABLE bit of o_Flags to indicate that the image is draggable.

OBJ_GImage - Iconic action gadget (image, but no 3D border).
I) Use the APSH_ObjData tag to specify the initial image to display.
SG) GA_LABELIMAGE
I) Set the APSH_OBJF_DRAGGABLE bit of o_Flags to indicate that the image is draggable.

OBJ_MultiText - Multiple line wrapping text gadget. Minimum width of fifty pixels, minimum height of thirty pixels.
I) STRINGA_MaxChars - Buffer size, in characters, to allocate.
I) STRINGA_Buffer - Application provided buffer.
I) STRINGA_UndoBuffer - Application provided undo buffer.
ISGN) STRINGA_TextVal - Displayed text.
IS) STRINGA_WorkBuffer
IS) STRINGA_BufferPos
IS) STRINGA_DispPos
IS) STRINGA_AltKeyMap
IS) STRINGA_Font
IS) STRINGA_Pens
IS) STRINGA_ActivePens
IS) STRINGA_EditHook

IS) CGTA_HighPens
I) CGTA_DisplayOnly
ISG) CGTA_Top
G) CGTA_Visible
G) CGTA_Total

OBJ_DirString - Text gadget with button for directory listing. Height is font height plus six.

OBJ_DirNumeric - Numeric gadget with button for directory listing. Height is font height plus six.

OBJ_boopsi - boopsi gadget or image.
I) Use the APSH_GA_Image tag to specify the name of the OBJ_boopsi object to use as the GA_Image object.
S) GA_Image
I) Use the APSH_GA_SelectRender tag to specify the name of the OBJ_boopsi object to use as the GA_SelectRender object.
S) GA_SelectRender

OBJ_View - Multi-column, multi-select view.
I) AOLV_Borderless - Place a border around the view?
I) AOLV_Freedom - Horizontal and/or Vertical.
I) AOLV_ReadOnly - List read-only?
I) AOLV_MultiSelect - List support multi-selection?
I) AOLV_ControlHeight - Height of the control panel.
I) AOLV_LabelHeight - Height of the column label bar.
IS) AOLV_UnitHeight - Height of each row. Use ~0 to use the screen font height plus one.
IS) AOLV_UnitWidth - Width of each horizontal unit.
IS) AOLV_List - List to display.
IS) GTLV_Labels - Same as AOLV_List.
ISGN) GTLV_Selected - Currently selected vertical row.
ISGN) AOLV_TopVert - Top vertical line.
ISGN) AOLV_VisibleVert - Number of visible vertical lines.
ISGN) AOLV_TotalVert - Number of total vertical lines.
ISGN) AOLV_TopHoriz - Top horizontal line.
ISGN) AOLV_VisibleHoriz - Number of visible horizontal lines.
ISGN) AOLV_TotalHoriz - Number of total horizontal lines.
GN) AOLV_View - View rectangle.

OBJ_MListView - Multi-column, multi-select scrolling list view. Same tags as the OBJ_View object.

The following objects are for images.

OBJ_Image - Display an image.
- I) Use the APSH_ObjData tag to specify the initial image to display.
- SG) GA_LABELIMAGE

OBJ_Column - Column image, that can only be used with OBJ_View and OBJ_MListView
- I) AOLV_ColumnWidth - Percent of total width. This is stored in image->ImageData.
- I) AOLV_Title - Title for the column.
- I) AOLV_Justification - Left or right justification.
- I) AOLV_FieldOffset - Byte offset of field within structure. If zero, then use the node->ln_Name field.
- I) AOLV_FieldType - Type of field, such as text, text pointer, long, etc.

The following objects are different types of borders.

OBJ_BevelIn - Pushed in 3D border.
OBJ_BevelOut - Pushed out 3D border.
OBJ_DblBevelIn - Pushed in 3D embossed border.
OBJ_DblBevelOut - Pushed out 3D embossed border.
- I) Use the APSH_ObjData tag to specify the initial image to display.
- SG) GA_LABELIMAGE
- I) Set the APSH_OBJF_DRAGGABLE bit of o_Flags to indicate that the image is draggable.

The following objects are basically different types of groups.

OBJ_Window - Window
OBJ_Group - Freeform layout group.
OBJ_VGroup - Vertical layout group.
OBJ_HGroup - Horizontal layout group.
OBJ_MGroup - Mutual exclusion group. Members can only be OBJ_Button or OBJ_Select.
OBJ_VFill - Vertical fill area.
OBJ_HFill - Horizontal fill area.

The gadget objects accept the following tags:

Message Handler Name

IDCMP

Tags

APSH_TextAttr - Text attribute to use.
APSH_NewScreen - NewScreen structure.
APSH_NewScreenTags - Tags to use when opening the screen.
APSH_Palette - Color palette to assign to the screen.

APSH_DefWinFlags - Default window flags.
APSH_WindowEnv - Used to describe a window environment.
APSH_NameTag - Name to assign to this window.
APSH_GTMenu - GadTools-style NewMenu array.
APSH_HotKeys - Keystroke array used to specify the function binding for any particular key.
APSH_NewWindow - NewWindow structure.
APSH_NewWindowTags - Tags to use when opening the window.
APSH_Objects - Array of objects in the Graphical User Interface. Translated into GadTools or Intuition objects, according to user preferences.

Extended Low-Level Message Handler Functions

<none>

Standard Functions

Activate

,
Restore the graphical user interface of an application to the state that it was in before receiving a DEACTIVATE command.

Deactivate

,
Shutdown the entire graphical user interface of an application.

Keyboard
KEY,WINDOW/K,GLOBAL/S,HOTKEY/S,PROMPT/S,CMD/F
Allows the user to bind a function to a key.

KEY
Specifies which keystroke to edit.

WINDOW
Specify the name of the window that the keystroke is active in. The window that the keystroke will be used in. Defaults to MAIN.

GLOBAL
Indicate that the keystroke applies to all windows for the application.

HOTKEY
Indicate that the keystroke is system-wide, regardless of active window.

PROMPT
Bring up a window that allows a GUI for selection of keystrokes and functions.

CMD
The command to assign to the keystroke.

Window
NAME/m,TITLE/k,OPEN/s,CLOSE/s,SNAPSHOT/s,ACTIVATE/s,
MIN/s,MAX/s,FRONT/s,BACK/s,ZOOM/s,UNZOOM/s,LOCK/s,UNLOCK/s
Allows the user to manipulate a window.

NAME
Specify the window to manipulate. Defaults to MAIN.

OPEN
Open the window.

CLOSE
Close the window.

TITLE
Specify the window title.

SNAPSHOT
Save the current window rectangle information to a file.

ACTIVATE
Activate the window.

MIN
Size the window to its minimum rectangle.

MAX
Size the window to its maximum rectangle.

FRONT
Send the window to the front of other windows.

BACK
Send the window behind all others.

ZOOM
Change the window box to its zoomed position.

UNZOOM
Change the window box to its unzoom position.

LOCK
Lock all input to the window.

UNLOCK
Unlock a previous LOCK.

WindowToBack
NAME
Sends the named window to back. Defaults to MAIN.

WindowToFront
NAME
Brings the named window to front. Defaults to MAIN.

Preferences

The AppShell will monitor user preference files. (I) indicates implemented, (NI) indicates not implemented.

palette.prefs - Colors to use in a custom or public screen (I). Automatically provides a pen spec.
pointer.prefs - Normal pointer (I).
busypointer.prefs - Busy pointer (I).
<name>pointer.prefs - Other pointers, such as mode (fill, text, etc) indicators, whenever appropriate (NI).
screenmode.prefs - Screen mode/type (I).
screenfont.prefs - Screen font, used for menus and window title text (NI).
sysfont.prefs - Font to use inside the windows (NI).
win.pat - Backfill pattern for windows (NI)
wb.pat - Backfill pattern for background window or screen (NI).
printer.prefs - Printer preferences (NI).
printergfx.prefs - Graphic printer preferences (NI).
<window title>.win - Window size and placement preferences (I).

Example

```
STRPTR Group_text[] =
{
 /* Padding */
    "NULL",

    /* Window title */
    "Window w/Groups",

    /* Checkmark labels */
    "Free Form",
    "Strip EOL Blanks",
    "TAB --> Spaces",
    "Translate CRs",

    /* Button labels */
    "_Use",
    "_Help",
    "_Cancel",

    NULL
};
```

```
/* The following tag arrays are GadTool tags for Window Environment
 * objects. */
struct TagItem check_tags[] =
{
    {APSH_GTFlags, (LONG) PLACETEXT_RIGHT},
    {TAG_DONE,}
};

/* This is the object array for the Group window. */
struct Object Group[] =
{
    {&Group[ 1], 0, 0, OBJ_Window, NULL, APSH_OBJF_LOCALTEXT, NULL,
     "Group", 1L, {0, 0, 200, 80},},

  /* Control buttons */
    {&Group[ 2], 1, 0, OBJ_HGroup, NULL, NULL, NULL, "Control", 0L,
     {8, -8, -16, 12},},
    {&Group[ 3], 1, 0, OBJ_Button, NULL,
     APSH_OBJF_LOCALTEXT | APSH_OBJF_CLOSEWINDOW, NULL, "Use",    6L,},
    {&Group[ 4], 1, 0, OBJ_Button, NULL,
     APSH_OBJF_LOCALTEXT,                        NULL, "Help",   7L,},
    {&Group[ 5], 1, 0, OBJ_Button, NULL,
     APSH_OBJF_LOCALTEXT | APSH_OBJF_CLOSEWINDOW, NULL, "Cancel", 8L,},

  /* Column of checkmarks */
    {&Group[ 6], 2, 0, OBJ_VGroup, NULL, NULL, NULL, "Misc", 0L,
     {8,  4, 180, 50},},
    {&Group[ 7], 2, 0, OBJ_Checkbox, NULL, APSH_OBJF_LOCALTEXT,
     NULL, "FreeForm", 2L, {0, 0, 0, 0}, check_tags,},
    {&Group[ 8], 2, 0, OBJ_Checkbox, NULL, APSH_OBJF_LOCALTEXT, NULL,
     "Strip",    3L, {0, 0, 0, 0}, check_tags, },
    {&Group[ 9], 2, 0, OBJ_Checkbox, NULL, APSH_OBJF_LOCALTEXT, NULL,
     "Tabs",     4L, {0, 0, 0, 0}, check_tags, },
    {NULL,        2, 0, OBJ_Checkbox, NULL, APSH_OBJF_LOCALTEXT, NULL,
     "Translate",5L, {0, 0, 0, 0}, check_tags, },
};

/* This is the Window Environment tag array for the Group window. */
struct TagItem Groupenv[] =
{
    {APSH_NameTag, (ULONG) "Group"},        /* name to give to window */
    {APSH_Objects, (ULONG) Group},          /* object list */
    {APSH_WinText, (ULONG) Group_text},
    {APSH_WinAOpen, OpenGroupID},
    {TAG_DONE,}
};

VOID
GroupFunc (struct Hook *h, struct AppInfo * ai, struct AppFunction *af)
{
    extern struct TagItem Groupenv[];

    HandlerFunc (ai,
             APSH_Handler, "IDCMP",
             APSH_Command, APSH_MH_OPEN,
             APSH_WindowEnv, (ULONG) Groupenv,
             TAG_DONE);

}
```

Notification Message Handler

The Notification message handler provides a way to watch a file and execute a function whenever that file is changed. This message handler is used to provide preferences for an AppShell application, and is therefore added to the application's message handler list automatically.

Message Handler Name

NOTIFY

Tags

APSH_NameTag, <file name> - Complete file name of file to watch.
APSH_CmdID, <function ID> - Function to execute when file is modified. Examine APSH_NameTag to get the name of the file modified.

Extended Low-Level Message Handler Functions

<none>

Standard Functions

<none>

Preferences

<none>

Example

Note that notification is started with the APSH_MH_OPEN command and can be stopped with the APSH_MH_CLOSE command. The Notification message handler will automatically terminate any outstanding notification requests when it does a shutdown.

```
\* Start notification on a file *\
VOID
WatchClipFile (struct Hook *h, struct AppInfo *ai, struct AppFunction *af)
{
    \* Start the notification *\
    HandlerFunc (ai,
             APSH_Handler,  "NOTIFY",
             APSH_Command,  APSH_MH_OPEN,
             APSH_NameTag,  "ram:temp",
             APSH_CmdID,    ClipFileChangeID,
             TAG_DONE);
}

/* Function that gets called whenever file is changed */
VOID
ClipFileChange (struct Hook *h, struct AppInfo *ai, struct AppFunction *af)
{
    struct TagItem *attrs = af->af_FE;
    STRPTR name;

    /* Get the name of the file that changed */
    name = (STRPTR) GetTagData (APSH_NameTag, NULL, attrs);
}
```

Simple Interprocess Communications Message Handler

The Simple Interprocess Communications Message Handler (SIPC) provides fast and simple communications between applications and tools.

The tool message handler uses the SIPC to tell tools and applications to shut down. The HyperText library uses SIPC to talk to an application that supports it.

Message Handler Name

SIPC

Tags

APSH_Port - Used to provide a name for the SIPC port. Defaults to basename plus _SIPC

APSH_Status - Recognizes:

APSHP_INACTIVE - When set, leaves the SIPC message handler inactive. Causes the message handler to reply to each incoming message with the return value set to RETURN_FAIL. When not set, causes SIPC to become immediately available.

APSHP_SINGLE - When set, uses the base port name as is. When not set, will append .# to the base port name, where # is incremented until it finds a unused name.

Extended Low-Level Message Handler Functions

AH_SENDCMD Sends the a command to the designated SIPC port. Understands the following tags:

APSH_NameTag - Specify the name of the SIPC port to send the command to.
APSH_PortAddr - Specify the address of the SIPC port to send the command to.
APSH_CmdID - Function for the destination application to execute.
APSH_CmdData - Data to pass to the destination application.
APSH_CmdDataLength - Length of the data block to pass. If zero or not present, then APSH_CmdData is assumed to be a TagItem array.
APSH_CmdString - Specify a command string to send to the destination application.

Standard Functions

<none>

Preferences

<none>

Tool Message Handler

The Tool Message Handler provides the application with the capability to run asynchronous processes. If an application allows multiple projects, the tool message handler offers a way for the application to start a new process for each project. It also allows tools relevant to the application to be run asynchronously. Such tools could be a color cycler, animator or real time display.

Message Handler Name

TOOL

Tags

APSH_Tool, <name> - Name to give to the process that is spawned. Required.

APSH_ToolData, <data> - Data to pass to the new process. Passes a pointer to the AppInfo structure by default. If you are starting a new AppShell application, then a tag list should be passed.

APSH_ToolAddr, <function> - Pointer to the function to spawn as a new process. Defaults to HandleAppAsync.

APSH_ToolStack, <stack size> - Stack size to allocate for the new process. Defaults to 4096 bytes.

APSH_ToolPri, <priority> - Priority to run the new process at. Defaults to zero.

Extended Low-Level Message Handler Functions

<none>

Standard Functions

<none>

Preferences

<none>

Handle Messages

Once the message handler list has been successfully initialized, the AppShell will go into its event processing mode.

While in the event processing mode, the AppShell will wait on the signal bits of all the registered message handlers. Whenever an event comes in, the appropriate message handler gets called. The message handler will then translate the events into function calls. This process continues until the user indicates that he or she wants to leave the application.

Shut Down Message Handlers

When the application is told to quit, the AppShell enters the shut down mode. While in shut down mode, the AppShell loops through the list of message handlers, calling the shut down routine of each message handler, until the list is empty.

A custom shutdown function can be specified using the APSH_AppExit tag, with the data being the function ID to call.

Implementing a Message Handler

What is a Message Handler?

A message handler, for the AppShell, is a module that converts events to function calls. For example, the Intuition message handler allows the programmer to specify a function to call whenever a gadget is selected, a window closed, etc.

Message Handler Requirements

Each message handler must provide the following low-level functions:

- Setup - Message handler initialization
- Open - Enable the message handler or one of its objects.
- Handle - Translate messages into calls to the AppShell function dispatcher.
- Close - Disable the message handler or one of its objects.
- Shutdown Shut down the message handler.

Setup

The Setup routine takes care of all the message handler initialization. It is invoked by the AppShell application with the APSH_AddHandler tag.

The following is code fragment showing some of the tags required to initialize an imaginary custom message handler.

```
/* Describe the custom serial message handler */
struct TagItem Handle_Serial[] =
{
    /* Entry point to the message handler */
    {APSH_Setup, setup_serialA},

    /* The handler is required for our application */
    {APSH_Rating, APSH_REQUIRED},

    /* End the tag array */
    {TAG_DONE,}
};

/* Application Environment */
struct TagItem Our_App[] =
{
    /* About the application */
    {APSH_AppName, (ULONG) APPNAME},
    {APSH_AppVersion, (ULONG) APPVERS},
    {APSH_AppCopyright, (ULONG) APPCOPY},
    {APSH_AppAuthor, (ULONG) APPAUTH},

    /* Other initialization tags go here */

    /* Add a custom handler (serial) */
    {APSH_AddHandler, (ULONG) Handle_Serial},

    /* End the tag array */
    {TAG_DONE,}
};
```

Open

The Open routine is responsible for enabling the message handler, or opening a message handler object.

The Open routine for the ARexx message handler enables the ARexx port, while the Open routine for the Intuition message handler allows the application to open a new or existing window.

The following code fragment shows how an application can call the Open routine of a message handler.

```
/* Enable the Serial message handler */
HandlerFunc (ai,
            /* Specify the name of the message handler to manipulate */
            APSH_Handler,    "SERIAL",

            /* Specify the Open routine */
            APSH_Command,    APSH_MH_OPEN,

            /* Specify the object to open */
            APSH_NameTag,    (ULONG) "Modem",

            /* Specify the function to invoke whenever a character is
             * received */
            APSH_CmdID,      ModemInputID,
            TAG_DONE);
```

Handle

The Handle routine processes all the messages coming into the message handler's message port, and converts them to function ID's to be dispatched.

The Handle routine should not be called by the application. It is invoked each time a message comes in the message port assigned to the message handler.

Close

The Close routine is used to disable a message or to close a message handler object.

For example, the Close routine will disable the ARexx port in the ARexx message handler, and is used to close windows in the Intuition message handler.

The following code fragment shows how an application can call the Close routine of a message handler.

```
/* Disable the Serial message handler */
            HandlerFunc (ai,
            /* Specify the name of the message handler to manipulate */
            APSH_Handler,    "SERIAL",

            /* Specify the Close routine */
            APSH_Command,    APSH_MH_CLOSE,

            /* Specify the object to close */
            APSH_NameTag,    (ULONG) "Modem",

            /* End the tag array */
            TAG_DONE);
```

Shutdown

The Shutdown routine is responsible for closing all open message handler objects, and then freeing all the resources obtained at Setup time.

The Shutdown routine must not be called directly by the application. It is called by the AppShell at shutdown time.

Message Handler Structures

Each message handler is describe to the AppShell by a MsgHandler structure. This structure is defined in *<libraries/appshell.h>*, as has the following format.

```
/* Message handler structure */
struct MsgHandler
{
    struct MHObject mh_Header;       /* Embedded MHObject structure */

    struct MsgPort *mh_Port;         /* Message port for handler */
    STRPTR mh_PortName;              /* Pointer to the port name,
                                      * if public */
    ULONG mh_SigBits;                /* Signal bits to watch for */

    /* low level message handler functions */
    WORD mh_NumFuncs;                /* Number of low level msg. handler
                                      * functions */
    BOOL (**mh_Func)(struct AppInfo *,struct MsgHandler *,struct TagItem *);

    STRPTR *mh_DefText;              /* Default text catalogue */
    APTR mh_Catalogue;               /* *** PRIVATE *** SYSTEM USE ONLY */

    LONG mh_Outstanding;             /* Number of outstanding messages */
    APTR mh_Extens2;                 /* *** PRIVATE *** SYSTEM USE ONLY */
}
```

The MsgHandler structure contains the following components:

mh_Header - Embedded MHObject structure which contains the base information for this message handler.
mh_Port - The message port that this message handler uses.
mh_PortName - A pointer to the constructed public port name for this message handler.
mh_SigBits - The signal bits for the message port.
mh_NumFuncs - Contains the number of low level message handler functions. Four are required: Open, Handle, Close, and Shutdown.
mh_Func - Pointer to an array of low level message handler functions. MH_OPEN, MH_HANDLE, MH_CLOSE, and MH_SHUTDOWN are required.
mh_DefText - Pointer to the default text table for this message handler. This is not required, in fact use of the global text table is strongly encouraged.
mh_Outstanding - Number of outstanding messages. If the message handler sends out messages, then it should use this field to track the number of outstanding messages.

The remainder of the fields are reserved for the AppShell.

One of the basic components of message handlers is the MHObject structure. This structure contains the fields necessary to maintain nested layers of message handler objects.

```
/* message handler object node */
struct MHObject
{
    struct Node mho_Node;                /* embedded Exec node */
    struct List mho_ObjList;             /* List of children objects */
    struct MHObject *mho_Parent;         /* Pointer to parent object */
    struct MHObject *mho_CurNode;        /* Pointer to current child object */
    ULONG mho_ID;                        /* Numeric ID of object */
    ULONG mho_Status;                    /* Status of object */
    APTR mho_SysData;                    /* Message handler data */
    APTR mho_UserData;                   /* application data */
    APTR mho_Extens1;                    /* *** PRIVATE *** */
    UBYTE mho_Name[1];                   /* Name of object */
};
```

The MHObject structure contains the following fields:

mho_Node - Embedded Exec list node structure.
mho_ObjList - List of objects that belong to this object. Each object must begin with a MHObject structure.
mho_Parent - Pointer to the parent MHObject for this object.
mho_CurNode - Pointer to the currently active MHObject in the mho_ObjList list.
mho_ID - Numeric ID assigned to this record.
mho_Status - Status of the record, such as enabled or disabled.
mho_SysData - Pointer to data maintained by the message handler.
mho_UserData - This field is provided to the application for its own purposes.

The remaining fields are private to the AppShell.

Message Handler Initialization

At initialization time, each message handler must initialize and return the MsgHandler structure. This structure is used by the event processor to manage each message handler.

The following is an example of a minimal message handler initialization routine.

```
struct DummyInfo
{
    ULONG di_Padding;
};

struct MsgHandler * ASM
setup_dummyA (REG(d0) struct AppInfo * ai, REG(d1) struct TagItem * tl)
{
    struct MsgHandler *mh;
    struct MHObject *mho;
    struct DummyInfo *md;
    ULONG msize;

    /* Calculate the amount of memory required */
    msize = sizeof (struct MsgHandler) +
            sizeof (struct DummyInfo) +
            (4L * sizeof (ULONG));

    /* Allocate the message handler structure */
    if (mh = (struct MsgHandler *) AllocVec (msize, MEMFLAGS))
    {
        /* Cache the pointer to the embedded MHObject */
        mho = &(mh->mh_Header);

        /* Initialize the node that is contained within the MHObject */
        mho->mho_Node.ln_Type = APSH_MH_HANDLER_T;
        mho->mho_Node.ln_Pri = APSH_MH_HANDLER_P;
        mho->mho_Node.ln_Name = "DUMMY";

        /* Initialize the object list, whether you use it or not! */
        NewList (&(mho->mho_ObjList));

        /* This is just message handler ID, so that the application can
         * tell which handler a message came from. */
        mho->mho_ID = APSH_DUMMY_ID;

        /* Get a pointer to the instance data */
        mho->mho_SysData = md = MEMORY_FOLLOWING (mh);

        /* The array of function pointers is set up so that
         * AppShell can dispatch messages as needed to the
         * proper message handler functions. */
        mh->mh_NumFuncs = 4;
        mh->mh_Func = MEMORY_FOLLOWING (md);
        mh->mh_Func[APSH_MH_OPEN] = open_dummy;
        mh->mh_Func[APSH_MH_HANDLE] = handle_dummy;
        mh->mh_Func[APSH_MH_CLOSE] = close_dummy;
        mh->mh_Func[APSH_MH_SHUTDOWN] = shutdown_dummy;

        /* Create the message port for Exec messages */
        if (mh->mh_Port = CreatePort (NULL, NULL))
        {
            /* Register the signal bit */
            mh->mh_SigBits = (1L << mh->mh_Port->mp_SigBit);

            /* Setup is complete, return the message handler structure */
            return (mh);
        }
        else
        {
            /* Set the error return to indicate that we couldn't create
             * the message port. */
            ai->ai_Pri_Ret = RETURN_FAIL;
            ai->ai_Sec_Ret = APSH_CLDNT_CREATE_PORT;
            ai->ai_TextRtn = PrepText (ai, APSH_MAIN_ID, ai->ai_Sec_Ret, NULL);
        }

        /* Cleanly exit */
        FreeVec ((APTR) mh);
        mho = NULL;
        mh = NULL;
    }
    else
    {
        /* Set the error return to show that we don't have any memory! */
        ai->ai_Pri_Ret = RETURN_FAIL;
        ai->ai_Sec_Ret = APSH_NOT_ENOUGH_MEMORY;
        ai->ai_TextRtn = PrepText (ai, APSH_MAIN_ID, ai->ai_Sec_Ret, NULL);
    }

    /* NULL return indicates error */
    return (NULL);

}
```

Message Handler Functions

Message handlers can add functions to the application's function table. The functions must conform to the AppShell function style (outlined in the Functions section). The functions must be added to the message handler initialization routine in following manner:

```
/* ID for the Stub function */
#define StubID MYHANDLER_ID+1L

/* function prototype */
VOID StubFunc (struct AppInfo *, STRPTR, struct TagItem *);

/* message handler function table segment */
struct Funcs handler_funcs[] =
{
  {"STUB",  StubFunc, StubID,},
  {NULL, NO_FUNCTION,}    /* end of array */
};

/* sample message handler initialization routine */
struct MsgHandler * setup_dummyA (struct AppInfo *ai, struct TagItem *tl)
{
        ...
        /* set up the signal bits */
        mh->mh_SigBits = (1L << mh->mh_Port->mp_SigBit);

        /* add the standard functions to the function table */
        AddFuncEntries (ai, handler_funcs);

        ...
}
```

The following is an example of how a standard function can access the message handler's data.

```
/* sample stub function to show how to obtain a message handlers' data */
VOID StubFunc (struct AppInfo *ai, STRPTR cmd, struct TagItem *tl)
{
    register struct MsgHandler * mh;
    register struct MHObject * mho;
    register struct myhInfo * md;

    /* get a pointer to the message handler */
    if (mho = (struct MHObject *)
        HandlerData (ai,APSH_Handler,"DUMMY",TAG_DONE))
    {
      /* get a pointer to the message handler data */
      md = mho->mho_SysData;

      ...       /* manipulate the message handler data */

    }
}
```

Functions can also call the low-level message handler functions. The following is an example of how a function can access the low-level message handler functions.

```
/* sample function to show calling low-level message handler */
/* functions                                                  */
VOID YourFunc (struct AppInfo *ai, STRPTR cmd, struct TagItem *tl)
{
  HandlerFunc (ai,
               APSH_Handler, "IDCMP",
               APSH_Command, MH_OPEN,
               APSH_NameTag, "MAIN",
               TAG_DONE);
}
```

This example would tell the IDCMP (Intuition) message handler to open the window that has the short name "MAIN".

Message Handler Open

Each message handler must have an Open routine that enables the message handler or one of its objects.

Here is an example of a minimal message handler Open routine:

```
/* sample message handler open function */
BOOL open_dummy (struct AppInfo * ai, struct MsgHandler * mh,
                 struct TagItem * tl)
{
  register struct MHObject *mho = &(mh->mh_Header);

  mho->mho_Status |= APSH_MH_OPEN;

  return (TRUE);
}
```

Multiple Message Handler Objects

Sometimes it is necessary for a message handler to track several different objects. A custom handler can achieve this by maintaining an Exec list, the MHObject field mho_ObjList, of all the available objects. A good example is the IDCMP message handler maintaining multiple windows.

Message Handler Handle

Each message handler must have a Handle routine which manages the messages for its message port. It is in charge of translating these message handler events into commands.

Here is a minimal example of a message handler Handle routine. This example uses the Simple IPC message structure as its form of communication:

```
/* minimal message handler */
BOOL handle_dummy (struct AppInfo * ai, struct MsgHandler * mh,
                    struct TagItem * tl)
{
    register struct MHObject *mho = &(mh->mh_Header);
    register struct SIPCMessage *msg;
    WORD FuncID;

    while (msg = (struct SIPCMessage *) GetMsg (mh->mh_Port))
    {
        /* Get the function number assigned to this message */
        FuncID = (UWORD) msg->sipc_Type;

        if ((FuncID != NO_FUNCTION) &&
            (mho->mho_Status & APSHF_OPEN) &&
            !(mho->mho_Status & APSHF_DISABLED))
        {
            /* Perform the function assigned to this key. */
            PerfFunc (ai, NULL, GetFuncName (ai, (ULONG) FuncID), tl);
        }

        /* reply to the message */
        ReplyMsg ((struct Message *) msg);
    }

    return (TRUE);
}
```

Message Handler Close

Each message handler must have a Close routine that disables the message handler or one of its objects.

Here is an example of a minimal message handler Close routine:

```
BOOL close_sipc (struct AppInfo * ai, struct MsgHandler * mh,
                    struct TagItem * tl)
{
    register struct MHObject *mho = &(mh->mh_Header);

    mho->mho_Status &= ~APSHF_OPEN;

    return (TRUE);
}
```

Message Handler Shut Down

When the ai_Done flag is set to TRUE and the ai_NumCmds field equals zero, the AppShell enters the shut down phase. This involves going through the list of message handlers and calling each handlers' MH_SHUTDOWN routine. This shut down is performed until the list is empty.

Each message handler's shut down routine is in charge of removing the message handler from the message handler list and removing its own resources. The message handler must make sure that all messages are replied before deallocating its port.

The following is a minimal message handler Shut Down routine.

```
/* minimal message handler shutdown routine */
BOOL shutdown_sipc (struct AppInfo * ai, struct MsgHandler * mh,
                    struct TagItem * tl)
{
    if (mh)
    {
        /* Remove the message handler from the list */
        Remove ((struct Node *) mh);

        /* Make sure there is a message port */
        if (mh->mh_Port)
        {
            /* Remove the message port */
            DeletePort (mh->mh_Port);
        }

        /* Free the message handler data */
        FreeVec ((APTR) mh);
    }

    return (TRUE);
}
```

Krieger, Zander & Partner GmbH
Localization, Book publishing, CBT-Development, Consulting

Founded: 1983
Employees: 25
Turnover: 2,5 Mio
Location: Munich

REFERENCES:
APPLE COMPUTER GMBH
APRICOT DEUTSCHLAND GMBH
ASHTON-TATE GMBH
COMPU-SHACK GMBH
COMPUTER ASSOCIATES
CO-TEXT GMBH
ERICSSON DEUTSCHLAND GMBH
IDEA ASSOCIATES GMBH
LOGISTICS DATA SYSTEMS GMBH
LOGON SOFTWARE GMBH
MAI DEUTSCHLAND GMBH
MANNESMANN TALLY GMBH
MARKT & TECHNIK AG
MICROSOFT GMBH
MIDAS GMBH
MIS GMBH
NCR GMBH
SIEMENS AG
SUN MICROSYSTEMS GMBH
TELENORMA DATENSYSTEME GMBH
TEXAS INSTRUMENTS GMBH
TRIUMPH ADLER AG
UNIVERSITÄT DUISBURG
XTREE COMPANY

Localization Services

1. Overview

Krieger, Zander and Partner GmbH offers European localization, especially into the key languages: German, French, Italian, Spanish, Portuguese (and/or Brazilian). Our full-service includes every working step from Translation Kit development up to the product ready for sale:

Consulting

The analysis of software and documentation leads to a judgement of chances and investment efforts for Localization projects.

- Estimation of source structures
- Tips for resource organization
- Test prerequisites
- Orientation towards target groups and product design

Product Development

Development of tools and concepts in order to realize Localization projects.

- Translation tools
- Testsuits
- Fixing the product profile

Adaptation

Adaptation instead of translation means customizing a product for the national market.

- Scientific and technical adaptation
- Cultural adaptation
- Didactic adaptation
- Corporate Identity

Testing

Software is checked for functionality, documentation for comprehensiveness.

- Functional test
- Usability test

Production

Localization as full-service includes the final product version ready for sale.

- Typesetting
- Filming
- Printing

Product Maintenance

Introduction to the market and further updating is supported.

- Secondary literature
- Training concepts
- Updates for versions 1.X

2. Approach

Our approach combines the advantages of the two relevant localization strategies:

- centralized project management and quality assurance
- local realization of translation

The centralized approach proves its effectiveness because of five reasons:

- Every localization project raises fundamental problems, e.g. how can the sort order be changed? umlauts, accents integrated? It is sufficient to solve these problems one time for all languages.

- One project management instead of many is cost reducing.

- Europeanwide acting companies need to have a European designed product.

- It is a major objective to get to an international FCS especially in key languages.

- Our clients deal and speak with just one partner.

The local realization of the translation is because of the highest quality, which just can be reached by a native speaker, who is based in the specific country.

3. Chances

Amiga software is characterized by an user-friendly, easy-to-use interface. This is because of its functionality and it should be also because of its language interface: Adaptation increases the comfort of using, helps to surmount the learning barrier of an unknown program. A worked through language interface is a standard especially for the target group, who uses high-end application programs and cannot be satisfied with American-German mixtures.

The actual situation offers the chance to develop an Amiga-wide glossary, which could be distributed to developers interested in translation. This would establish a standard leading to an homogeneous appearance of Amiga Software.

It should be taken into consideration that a terminology is also a marketing instrument to force the success of a product on a national market. This is not only applicable for software but also for documentation, which is more but a technical information: a presentation of the product.

4. Software Localization Procedure

According to our experiences the working effort to localize complex product comes up to the effort of a multiple translation. As an average figure we calculate with three translation cycles for Software. Sometimes a lot more integration/compilation cycles are necessary, but - spoken idealized - it does not change anything in the specific character of each phase:

- The First Cycle Is Necessary To Work Out A Basic, A Literal Translation. This Translation Is Integrated And Given Away To Test. The Test Is Checking For Language Related Problems: Is The Translation Adequate To Its Context? Adequate To Target Groups? Correct? Such Leads The Test To A Revision Of The First Translation. About 50% Of The Total Translation Time Is Needed For The First Phase.

- The Second Cycle Begins With A Bug-Fixing, Correcting The Problems Found During The First Cycle By Test. This Revision Is Integrated Once Again And Given Away To Test. The Tester Now Looks Especially For The Screen Appearance Of The Translation. Are Messages, Options And Help Texts Completely Shown And Aligned? Are The Boxes Fit? Are System Values Displayed At The Right Place? Spelling Correct? The Second Cycle Needs About 33% Of The Total Translation Time.

- The Third Cycle Also Begins With A Bug-Fixing. The Integration Is Performed And The Product Is Getting Into Our Certification Procedure, Which Checks The Fixing Of Reported Troubles And Bugs. After This Last Cycle The Product Is Delivered To Our Client. It Needs About 17% Of Total Translation Time.

5. Adaptation Of Documentation

Once again we would like to point out, that a literal 1:1-translation is not an objective of translation. An adaptation to country specific conditions has to be performed, especially in didactics and addressing of target groups. This is worked out during a conception phase, in which the glossary, the graphic conception, the project plan etc. is drawn up. The start-up of the realization phase is a meeting, to give our client an idea, what the product will look like.

A documentation text has to pass the following cycles until it is delivered to our client:

- literal translation

- expert lecturing, editing, partial rewriting - The editor particularly takes the text as a German text into consideration, edits and partially rewrites it, where more explanation is needed. The result is - spoken idealized - a text, which cannot be identified as a translated one.

- DTP-production - The result of DTP-production are postscript files ready to be printed.

- usability test - The usability test checks the consistency of Documentation and Software. The tester is recapitulating every description and working practically on a computer.

- proof-reading

- format control - This is to be a last check for appearance on paper.

- filming, printing - This last step can be performed on demand.

6. German Product Maintenance

As a publisher of book series Krieger, Zander & Partner GmbH stays active to support the products which were localized. The know-how acquired during the localization process is transposed into suitable books, which also help a program to become successful.

As a service for developers Krieger, Zander & Partner GmbH can release a set of recommendations, which are useful to minimize the translation effort. This includes hints for the code structure as well as the distribution of a Commodore glossary. In general our know-how can be given away in terms of consulting.

Developers who want to sell on the German market and whose program can be recommended by us can be supported with a contact to German distributors.

7. Costs

In general costs are project and language specific. However to give a rough idea on our pricing the following numbers are applicable:

- Documentation Running Through The Procedure Described Above Is About $60.--/Page (Incl. Dtp-Production)

- Software Running Through The Cycles Described Above Is About $9.-- /Message And $50,-- /Text Screen (Excl. Integration Procedures)

8. Software Translation

discussed from a more technical point of view

8.1. Areas Which are Affected by Translation:

8.1.1 Text: This is the most obvious part of translation. But not all US products support the 8 Bit character set used by most of the foreign countries. Be aware that foreign languages usually need 20% to 50% more space to express the same idea. If your screen is already fully packed in the English version, it may be very hard to translate without being forced to abbreviate a lot. Don't do dynamic message concatenation, otherwise some combined message may not be translatable grammatically correct at all.

8.1.2 Folding and Collation sequences: The way strings are case converted and sorted differs between the individual countries. If you do any case conversion or sorting in your program which applies to accented characters you might have to set up conversion and sort tables. E.g. upper case conversion in C-Compiler libraries (toupper) does not work for accented characters.

8.1.3 Keyboard input: Foreign keyboards don't support all the characters of the US-type keyboard. You may not be able to enter some of the characters of the US keyboard. Your program has to handle the additional characters of the foreign keyboard properly. Accented characters should be usable as hotkeys (e.g. ALT-Ü). Again case conversion may give troubles (toupper).

8.1.4 Numbers: Numeric separators are quite often different. Many countries use the comma as the decimal point and a dot as thousands separator. Currency symbols and placement is another item of 'translation'. Currency symbols may consist of more than one character. Placement may be before/after with/without blanks. Date and time formats vary usually between the countries (like names for days and months). Non English speaking countries usually use 24 hour time format without am or pm. (e.g. effects display time at some screen location) Literals like 'Y', 'N', 'T', 'F' should be translatable.

8.1.5 Printer issues: Printer drivers may not work properly with the local version of the same printer. Some printer drivers don't support accented characters or create them by overprinting which generally not acceptable (e.g. ö is printed as o BS ") Paper and label sizes vary from country to country.(e.g. number of lines on a page in Germany is 72 compared to the US 66)

8.1.6 Telecommunications: Transmitting text over serial ports with a COM setting of 7 data bits gives troubles with accented characters. A country specific mapping table can solve this problem. As Telecommunications are specific for each country, no US only items should be removable in translated versions. (e.g. A menu item providing quick access to MCI mail doesn't make sense in Europe)

8.1.7 Disk space: As translated versions of a product usually grow a little bit in size, make sure that there is enough spare space on the disks.

8.1.8 Delimiters: Using comma as delimiter for separating individual items collides with the comma used as decimal point. It's safer to use the semicolon as item separator.

8.1.9 Macros: If macros work directly with the hotkeys of thc program, a existing macro won't work in a translated version.

8.1.10 Import/Export and other reference to third party software: File formats and file extensions may vary from the US version of third party products. (File viewers...)

8.2. TK Coverage:

In general a TK has to give the translator all necessary tools to create a translated version of a program.

TK Requirements:

Cover all necessary parts of translation (can be with different tools). See items discussed in previous section.

Give enough message context information to be able to do a correct and optimal translation. This can be done by some comment on each message about the area where the message shows up, a short explanation, the screen location, possible concatenation.

Specify 'limits' of translation, like minimum and max. message length, allowed characters, length of an inserted field, handling of leading or trailing blanks (e.g. "Give destination path: %s", how long will the test inserted by %s max/min be).

Provide 'instant reference' to corresponding English message/screen, preferably by function key.

Don't give permission to do any changes to source code or related parts. Goal: the translator should be able to do every possible change with the TK-tools without influence on the functionality of the program.

Provide tools to create a complete translated version of the product including creation of the disk set (Integration).

Specify test scripts and provide the translator with tools do test the translated product.

8.3. TK Methods and Tools:

8.3.1 Translation tools: For translation of menu items and messages a 'database method' suits very well, if text alignment is done automatically (box size and position...) In the 'database method' any translatable item (like messages, menu items, literals...) is stored in one record of a database. Together with the item to translate all necessary additional information like max. message length, context, the English message... are stored in the same record. A small 'screen mask' program (written in the database language) allows the translator to translate the message, view the English message, search in the English and translated text, print English/translated messages.... After translation a second program creates from the database a file in a format suitable for the integration process (e.g. *.h file). A third program merges new records in the translated database if an update of product occurs. This way the translator has only to care about new messages in the database. All this programs are not complicated and easy to develop.

Translation of all other parts like collation tables, numeric formats... should be done with either special tools or the database method. Readme or any other plain text files can be handled by normal text editors or wordprocessors.

8.3.2 Integration methods: 'Zap method': In the executable or overlay file an area is reserved and preceded by a magic string. The translated messages are preprocessed by the appropriate tool and then patched into this reserved area by a patch program which searches for the magic string and patches the data. This method is very easy to implement but not very space efficient. Access to the individual message is simple if the message length is kept the same as in the English version. This is achieved by allocating some spare memory in the English message. (e.g. '\0' padded character fields)

Link method: Create new binary file by linking the compiled messages with code in *.obj. Very efficient method, but more difficult to handle for the translators. In addition some of the tools have to be given to the translator. Memory layout of the new *.exe or overlay file will change which may have impact on functionality.

Resource file method: This method is similar to the 'Zap method' but the resource file is read at runtime. If done consequently, it avoids the waste of memory of the 'Zap method' and doesn't even require any compile or link of the product.

Pre-Mastering and Mastering for CDTV

by Mike Kawahara

1.0 Introduction

Preparation of titles for the CDTV player presents new challenges to the developer. Combining graphics, animation, text, digitized IFF sound samples and CD audio (or CD-DA) samples in the same application provides endless opportunities for creation. Multimedia applications for CD ROMs also require new production methods. This document will describe the various steps and techniques necessary for taking an application which runs on an Amiga SCSI hard disk and transferring it to a CD, eventually adding CD-DA sound along the way, and having that CD replicated.

Finally, we will discuss how the C-Track Emulator can simplify and expedite this process.

We will assume that the application runs correctly on an Amiga 2000 under Workbench 1.3. The code, data, and IFF sound files are on an AmigaDOS SCSI hard disk. The CD-DA sound tracks are on a DAT tape or, alternatively, on a 3/4" U-matic tape.

A list of pre-mastering and mastering service centers is attached.

2.0 Overview of the Pre-Mastering and Mastering Cycle

Some developers may prefer to avoid the pre-mastering job. It is possible for you to send your AmigaDOS SCSI drive to a pre-mastering center directly and have them do the pre-mastering for you. They will charge you more for that service, and you will lose some control over the optimization of your ISO disk layout with this method.

We can divide the process into a number of steps. Here follows a brief description of the steps. Each step will be explained in detail in Section 3.

2.1 Preparing the Tools

Before starting, you should verify you have all the hardware necessary to complete the job as well as the appropriate software tools.

2.2 ISO Control File Builder

The ISO tool scans the directory of your AmigaDOS drive and builds an ASCII control file with the disk structure and file hierarchy. You may wish to modify and optimize this control file manually.

2.3 Creating an ISO Image

The Buildtrack tool reads a control file and creates a binary ISO image file. This file can be stored as a file on an AmigaDOS disk. Alternately, you can use the same Buildtrack tool to create a block-for-block ISO image on another SCSI hard disk. The entire SCSI disk will be used--you cannot create a block-for-block image on a second partition of an AmigaDOS drive.

2.4 Creating a Test Disc

The ISO image file must now be transferred to a pre-mastering system, such as a Meridian CDPro. The SCSI disk connects directly to the Meridian. The Meridian then prepares a test disc, using a write-once CD-ROM drive such as a Yamaha PDS. The Meridian can also accept DAT tapes or 3/4" U-matic taper for CD-DA.

2.5 Testing

The Pre-Mastering Center will return a test disc to you. You should verify this disc's operation in a CDTV through vigorous testing.

2.6 Mastering and Replication

Satisfied with your testing, you tell the Pre-Mastering Center to prepare a pre-mastered tape. This tape, either 9-track or 8 mm, containing a logically formatted ISO 9660 image, is sent to the replicator of your choice. The replicator creates a master stamper that is subsequently used to press the CDs. Replicators also provide packaging services and ship your CDs, in jewel cases or long boxes, to you.

3.0 Description of the Production Cycle

3.1 Preparing the Tools

3.1.1 Hardware

The basic platform necessary to the developer for production is fairly simple: an Amiga with at least 3 Mo of RAM and one or (preferably) two SCSI hard disks.

Amiga: You can use either an Amiga 2000 (or 2500) or an Amiga 3000. Remember that CDTV currently contains Workbench 1.3 in ROM, and your application must not contain any WB 2.0-specific code. If you have an A2000, make sure you use an A2091 controller. The A2090 and A2090A controllers may have difficulty in controlling more than one SCSI drive.

SCSI hard disk: We have verified and recommend using the following makes of high-capacity SCSI drives:

Approximate
Retail price

Seagate ST4766N600 Mbyte formatted capacity$2400
Seagate ST2502N410 Mbyte formatted capacity$1300

3.1.2 Preparing the SCSI Drives

Up to 6 SCSI devices may be daisy-chained behind one A2091 controller. For the physical connection, follow the instructions in the 2091 User Manual. Be sure to set the device number on each drive, avoiding 2 drives with identical numbers.

Use the HD tool box to partition your drives, and verify that they are correctly recognized.

Buildtrack lets you create an ISO image file on two different devices. You can create a block-for-block image on a SCSI hard disk, or you can store the ISO image file on your AmigaDOS hard disk as an AmigaDOS file.

3.1.2.1 The block-for-block method will usually be chosen by developers who do not have direct access to CD mastering equipment. Once the block-for-block image is created on a hard disk, you simply send the hard disk to a pre-mastering center. They will read the data onto their mastering equipment and prepare a test disc.

If you choose this method, the disk must be connected as SCSI device 5.

3.1.2.2 The AmigaDOS file method allows you to store your ISO image as an AmigaDOS file on any AmigaDOS device. This method may be preferred in the following situations:

- if you are in a networking environment, with the mastering equipment connected into the net. You can then transfer the image file to the mastering equipment via the network.

- if you are physically near mastering equipment, you can transfer the image file to the mastering equipment via serial or parallel lines.

- if you want to keep more than one ISO image file in your system, for testing or control purposes.

3.1.3 Software tools. The ISO Dev Pak diskette contains all the necessary tools to create an ISO 9660 image.

3.1.4 Preparing your application disk. You must include a few important items on your application disk:

a. devs: directory. To avoid the "Workbench 1.3 Copyright Commodore" message at boot time, include a system configuration file in the devs directory with the four workbench colors all set to black.

b. c: directory. Copy the files "rmtm" and "CDTVPrefs" from the c: directory of the ISODevPak diskette to the c: directory of your application disk.

c. root directory. Copy the file CDTV.TM from the ISODevPak diskette to the root directory of your application disk.

d. s:startup-sequence. Add the command "rmtm" at the beginning of your startup-sequence in the s: directory of your disk. (If you include the Setpatch command, put rmtm just after Setpatch.) You must also add the command "CDTVPrefs" in your startup-sequence. This command reads the Prefs in the CDTV non-volatile RAM, and centers the screen accordingly.

3.2 ISO Control File

The ISO Control File Builder automatically scans your AmigaDOS drive and builds a control file for the actual CD image builder. This control file is used by BuildTrack to build a CD-ROM track image on an AmigaDOS hard disk, or block-for-block disk image on a SCSI disk.

The Control File Builder will build your new ISO file based on the pathname you provide. If, as an example, you wish to build the entire drive, you would simply type ISO <drive name:>. If you want to build only a directory, just give the full path that you wish to build, i.e. "ISO dh0:mydirectory". Always keep in mind that the Control File Builder will build an exact image of an ISO CD. The Control File Builder will construct the ISO directory for you and will optimize that directory structure for you.

3.2.1 Running ISO

Here are the steps involved in creating a an ISO control file:

1: BOOT with the ISO Devpak diskette

2: Open a Shell

3: type cd RAM:ISO2 <RTN>

4: type iso (source path) <RTN>

```
Volume ISO9660      :(Destination path)   <RTN>
VolumeIdentifier    :(name i.e.  Myvolume)  <RTN>
VolumeSetIdentifier :(name i.e.  Volume 1)  <RTN>
PublisherIdentifier :(your company name)   <RTN>
```

The *source path* is the pathname describing where your application resides. It may be a full drive (i.e. "dh0:") or just a single directory (i.e. "dh0:mydirectory")

The *Destination path* is the pathname and filename where the ISO image will be stored. If you are connected directly to the CD Mastering equipment, the Destination path is normally DR1: In that case, you should skip section 3.3.2.

Alternatively, the Destination path can be any AmigaDOS filename (i.e. DH1:myfile). In this case, the file may be transferred to the CD Mastering equipment via a network, or serial or parallel lines.

The *VolumeIdentifier* is a text string with the name of the ISO volume to be created. You may enter any name you wish.

The *VolumeSetIdentifier* a text string with the name of a volume set. You may enter any name you wish.

The *PublisherIdentifier* is a text string with the name of your company.

The VolumeIdentifier, VolumeSetIdentifier and PublisherIdentifier will be written in track 0 of the ISO volume.

Caution !!!!!

ISO should be run in RAM DISK, running it on the source drive would, of course, modify the disk you are building as it is being built.

ISO will create a file called "controlfile" in your RAM:ISO2 directory. Don't re-boot yet.

3.2.2 Optimizing the controlfile.

The controlfile is ASCII text, so you can read it or edit it if you wish. You may also wish to relocate where files and/or directories reside on the ISO CD-ROM, to optimize performance.

By default ISO sorts the files in a directory in ascending order by file size. This is generally the optimum order for the CDFS. However, if your application would operate faster with the files in some other order, you are free to change the order of the files.

For example, if file Pic.1 is to be loaded, then the sound files Noise.1, Noise.2 and Noise.3 are to be played sequentially, you might choose to put all 4 files in one directory, in that order.

What you must NOT do is change the directory in which a file resides. If you wish to do so, you must physically change them on your source hard disk, then run ISO again. If you move a file from one directory to another manually, the BuildTrack program will return a "file not found" error.

3.3 Creating An Iso Image

Two options are available for creating ISO images: a single track image, or a block-for-block disk image. First let's review briefly the format of an ISO disc.

3.3.1 ISO Disc Structure

An ISO 9660 standard CD-ROM is divided up into a number of variable-length tracks. These tracks are recorded in a spiral, from the center of the CD outwards. The spiral is 3 miles long on a full disc.

Track 0, known as the "subchannel", is reserved for the system. It contains information such as:
- the type of system on which the disc can boot
- the Table of Contents of the disc (TOC). The TOC describes the number of tracks on the disc, their location on the disc, and if the track contains CD audio or data.

Track 1 is the usual location for all data files. Here are stored your program's code, graphic files, IFF sound files, animations, etc. Again, this track is as long as needed to contain all your application's data.

Tracks 2-99 are for CD Audio (CD-DA) or other data tracks. These tracks are also of variable length. You may determine how to store your CD-DA information on these tracks. You may put one "song" or voice narration per track. Or you can have one long track with numerous samples. The CDTV Device Driver provides means for playing an entire track, or a certain portion of a track, beginning at a precise location (mm:ss:ff, where mm is minutes, ss is seconds, and ff is frames (75 frames per second)).

3.3.2 Preparing the Hard Disk

If you are connected directly to CD mastering equipment, you can skip this step, and go directly to 3.3.3.

Now that your controlfile is in RAM: and has the files in the order you prefer, you are ready to prepare a hard disk to receive the block-for-block ISO image. **WARNING: This step will erase the contents of the hard disk connected as SCSI device 5!**

From the CLI, type:

```
run bytedrive scsi.device 5 <RTN>
```

This will re-direct the output from BuildTrack to device 5. The bytedrive utility can only write to device 5.

Note that bytedrive requires both your application disk and the target disk to be attached to the same A2091 controller.

Now from the CLI type:

```
mount dr1:<RTN>
```

This mounts a special volume, dr1: which is addressed by BuildTrack.

3.3.3 Creating an ISO Image File

The Buildtrack utility lets you create an ISO disc image file of one track on Amiga disk, or on a network disk. The process is simple. While still in the RAM:ISO2 directory, type:

```
buildtrack controlfile -w
```

This will start the image process and will save the image file to the destination path you specified with the ISO command. (see section 3.2, step 4)

The -w switch will turn off the ISO filename warnings. These warnings are generated because the CDTV file system (CDFS) accepts any standard AmigaDOS file name. Many of those names are not part of the ISO 9660 standard, but are CDTV-specific extensions to the standard. For example, the ISO standard limits file names to 8 characters, and their extensions to 3. (Uncanny resemblance to another operating system, no?) BuildTrack will provide warnings about fatal errors, however.

Upon completion, BuildTrack will generate an ISO image file, either on your destination AmigaDOS hard disk, or a block-for-block image on SCSI device 5.

3.3.4 Running Fixtm

In order for your CDTV application to boot directly, you must include a special file called CDTV.TM in your root directory. This file is provided on the ISODevPak diskette.

The fixtm utility, on the same diskette, is used during pre-mastering to update the ISO image file with key information. You should run fixtm when you have finished the buildtrack utility.

Fixtm operates on both block-to-block hard disk images and on single track image files.

a) If you have created a block-to-block image on SCSI device 5, or on device 0, fixtm will read that image, and make sure it is a valid ISO image. It will then ask you the following question:

"Device <n> contains an ISO image, volume name <yourvolumename>. Update this image? (Y/N)"

Type Y to confirm, and fixtm will update the image to render your application directly bootable.

b) If fixtm finds no ISO image on SCSI device 5 or 0, it assumes you have created an image file on an AmigaDOS device. It then asks for the name of that file:

"Name of AmigaDOS image file to be updated:"

Here you should give the same destination path and filename as in section 3.2.1 step 4.(i.e. "DH0:myapp/imagefile"). The program will then update that image to render your application directly bootable.

3.4 Cutting a Test Disc

The ISO image must now be transferred to a pre-mastering facility to cut a test disc.

3.4.1 Media for Data Transfer

Currently two methods exist for physically transferring the image to the disk of the pre-mastering system: sending a SCSI hard disk drive, and a tape backup.

a) The most direct method is to ship the SCSI drive to the pre-mastering center. This method, while primitive, is fast and convenient. We have found that SCSI drives are surprisingly robust. If properly packaged, they can be sent via DHL, FedEx or equivalent and are rarely damaged en route.

The pre-mastering center simply plugs in your SCSI drive, block reads the data across to the Meridian's drive, and returns the drive to you along with the test disc. You will have to specify how many bytes of data are on the drive.

b) Alternatively you may back up your application to tape. We have tested the A3070 SCSI tape drives from Commodore, and they work well. Their major limitation: they only accept tapes of 150 Mbytes maximum. If your application goes beyond 150 M, you will have to split up the image, using the BRU utility.

The A3070 can only be used to store a track image. No software is currently available to transfer the block-for-block ISO image from a SCSI drive to the A3070.

We have been unsuccessful with Exabyte drives. Our testing has found the hardware to be unreliable. However, Exabyte is one of the most popular formats for European replicators.

We are investigating solutions to back up to a DAT drive, but have yet to find an inexpensive and reliable system. Any experience from developers would be greatly appreciated.

No matter what format you choose, make sure you indicate the total size of the ISO file you send to the pre-mastering company. Otherwise they will not know if they have transferred the entire file to their pre-mastering equipment.

3.4.2 Including CD-DA

If your application uses CD-DA, now is the time to supply the CD-DA data to the pre-mastering company. The formats accepted may vary from DAT tape to 3/4 inch Umatic tape.

When you create your CD-DA soundtracks, you should have them recorded at 0 decibels. The CDTV player is designed to attenuate the level of the Amiga sound, so that an Amiga sound played at maximum level will be just as loud as a 0-decibel CD-DA recording.

You should include SMPTE time code data on your CD-DA tape. This is obligatory if you wish to use the PlayMSF routines in your application, as they address the SMPTE codes directly.

See section 3.6.1 and Appendix B for more information on CD audio tapes.

3.5 Testing

Now comes a crucial step -- the verification of your test disc, before replication in quality.

3.5.1 "Gold" Disc

The pre-mastering facility will prepare a write-once CD, sometimes called a Gold Disc. This disc can be used for testing purposes.

You may want to ask for more than one copy of the Gold Disc, to test in multiple sites.

3.5.2 Testing

We strongly suggest that you test your application thoroughly. Points to consider:

- Try to run through all possible paths to your code and data.
- Load times. Verify that loading of graphics and audio is properly co-coordinated. Make sure you don't start playing an audio file too soon, or too late.
- Animation/video. Verify your animation or video sequences. If things are too slow, you might consider using alternate compression routines.

3.5.3 In Case Of Trouble

If you find problems, you will have to modify your code, then prepare another ISO image. Return to step 3.2. Do not pass GO. Do not collect $200.

However, all is not lost. If you have transferred all your data, IFF sound files, animations, code, and even CD-DA data onto the test disc, and discover timing problems or other bugs in your code, DON'T THROW AWAY YOUR TEST DISC! You can modify your code on your A2500 development system, and recompile there. Then transfer your code to a floppy diskette.

Now connect an A1011 floppy disk drive to the rear of your CDTV reader, insert your test disc in the CDTV, insert your floppy disk, and boot from the floppy. Your program on the floppy can access the data on the CD. It can play CD-DA tracks on the test disc as well. Thoroughly testing your application in this environment can avoid extra trips to the pre-mastering center for test discs.

3.6 Mastering And Replication

Having thoroughly tested your application on a write-once CD, you are ready for the final steps: mastering and replication.

3.6.1 Producing The Master Tapes

While it is possible to create a "master stamper" directly from a write-once CD, most replication centers prefer to receive a 9-track tape for input. This tape is prepared by the pre-mastering facility that made your test disc.

The master tape is an ANSI-labeled, 9 track, 1/2 inch tape containing an image of your applications code and data. The file may be split up over 2-4 reels, if the application cannot reside on one.

3.6.2 Preparing Cd-Da Tapes

If your application includes CD-DA audio, separate CD audio tapes will be created. CD Audio master tapes preferably have contiguous files with no files broken up across reels. Each 9-track tape holds approximately 138 Mbytes. You can write multiple files to each tape, but all the files must be contiguous.

The replicator will need to know where to place the tracks on the CD. The Track Sheet tells the replicator where to locate each track on the disc. The track sheet should also show the audio gap time between each track. The default gap time between tracks is 2 seconds. CD-ROM standards require that a 2 second gap be placed between DATA and AUDIO tracks, but there is no gap requirement between two AUDIO tracks. The Track Sheet will tell the replicator if you want a gap time different from the default 2 second time. Any gap time, even 0 seconds, is valid between two AUDIO tracks.

The Track Sheet can also tell the replicator where (in CD time) to start the tracks. Please refer to the sample Track Sheet in Appendix B.

3.6.2 Replication

Upon receipt of the master tapes, the replicator begins work. A detailed description of the numerous steps involved is beyond the scope of this document. But here is a brief overview.

(1) The tape's image is transferred to a large hard disk system. That system adds extra information, such as the synch pattern, the header, and error detection and correction data. Each sector is thus expanded from 2048 bytes to 2352 bytes.

(2) The sectors are transferred to the Laser Beam Recorder. This signal-processing rack and recorder adds error correction and subcode data, and performs the low-level encoding. The subcode data tells the CD-ROM drive where the head is located, independent of the sector data. The Recorder then exposes the master disc, a glass disc coated with a very thin layer of photoresist.

(3) The master is developed, and the photoresist which was exposed to the laser beam is etched away, leaving the pits. It is then coated with a thin layer of silver. The master can now be read, and is carefully tested on a special player to ensure quality.

(4) A "stamper" is created to actually press the CDs. Nickel is electroplated onto the silver surface of the master. The nickel shell is then separated from the glass master, creating a mirror image of the master.

(5) The stamper is placed on a molding machine. Molten polycarbonate is pushed between it and a mold. This polycarbonate is given an aluminum reflective layer, and a protective coating is added. It is now a CD.

(6) A label is printed on the CD, either via silk screening or pad printing. The CDs are put through a final quality assurance inspection. Finally, the replicator packages the CD, in a jewel case or a long box, and shrink-wraps the product.

4.0 The C-Trac Emulator

The C-Trac emulator, soon to be available from ICOM, will enable the developer to shorten his development cycle and reduce his pre-mastering costs.

4.1 Hardware And Installation

The Emulator is described in detail elsewhere in the Developer Notes. It is a board that plugs into the slot of an Amiga 2000 or Amiga 2500. It is connected via a ribbon cable to the motherboard of the CDTV.

The developer must format a hard disk under Cross DOS. This Cross DOS disk will be used to store the ISO 9660 image of your data necessary for the emulator. You can also store regular AmigaDOS files on the Cross DOS drive. The AmigaDOS file system will mount the Cross DOS drive, with its own file system, and you can use standard AmigaDOS commands to manipulate and access the data. Thus the same physical hard disk can hold both the ISO image and the AmigaDOS data and code.

4.2 CD-DA Files

Furthermore, you can store CD-DA data on the Cross DOS drive. The CD-DA files are seen as normal files by the file system. You can capture the files with tools currently available only on PCs or MacIntoshes. We are waiting for CD-DA sound digitizers from two Amiga developers. We will keep you posted when they are available.

4.3 Emulating a CD

When a special command is typed, the emulator activates. It looks for the s:startup-sequence file on the ISO image of the hard disk, loads that file into the memory of the CDTV, and starts to run.

Whenever the CDTV wants to access the CD, the emulator board intercepts the call. It reads the data from the Cross DOS disk in the Amiga 2500, emulating the seek time of a CD. It also slows down the transfer rate from the disk to the memory of the CDTV, to approximate the 150 Kbytes/second used on the CDTV.

The CTrac system allows you to test your application in an environment which provides response times very similar to those of a test disc. It can save you thousands of dollars by avoiding repeated cuts of test discs.

Appendix A

Names and addresses of Premastering and Mastering centers

The following companies pre-master, master, and/or replicate CD-ROMs. They should be contacted individually for specialties and pricing information:

3M Optical Recording Department
3M Center 223-5S-01
St. Paul, MN 55144-1000
tel. (1) 612-736-5399
fax (1) 612-733-0158
Sales contact: V.R. (Dick) Pendill
Technical contact: Jerre R. Lee tel. (715) 235-5567

Territories served: North and South America, Europe
Preferred Source format: 8mm Exabyte, 4 mm DDS DAT
Terms of Business: Net 30 days

Advanced Media Group, Ltd.
P.O. Box 1623
Lancaster, PA 17603
tel. (1) 717-392-6533
fax (1) 717-392-0532
Sales contact: Stan Caterbone
Technical contact: Stan Caterbone/Mike Hess
Territories served: North and South America, Europe, Far East
Preferred source format: Tape, hard disk. All media accepted upon prior approval.
Terms of Business: Upon invoice; Net 30 with prior approval
One-off service: available
Pre-mastering and CDTV ISO building service: soon available

Digipress
2516 River Bend Drive
Louisville, KY 40206
tel. (1) 502-895-0565
Contact: Dennis Oudard

In Europe, contact:
Digipress
10 rue de Paris
78100 Saint-Germain-en-Laye
France
tel. (33) 1-30-61-11-00
Contact: Marc Deflassieux

Digital Audio Disc Corporation (Dadc)
1800 N. Fruitridge Ave.
Terre Haute, IN 47804
customer service tel. (1) 812-462-8192
customer service fax (1) 812-466-2007
Sales contact: Bob Hurleytel. 603-595-4331
fax 603-595-4310
Technical Contact: Cliff Brannon tel. 812-462-8286
fax 812-466-9125
Territories served: US, Europe, Far East
Preferred source format: 9 track tape, MO, 8mm, 4mm, CD WO
Terms of Business: 1% 10 Net 30
One-off service: available
Pre-mastering and CDTV ISO building service: soon available

Discovery Systems
7001 Discovery Boulevard
Dublin, OH 43017
tel. (1) 614-761-2000
fax (1) 614-741-4258
Sales Contact: Greg Tiller
Technical Contact: Alex Deak, Customer Service

Territories served: North America, Europe, Australia
Preferred Source format:9 track or 8mm tape. Call for additional options.
One-off service: available
Pre-mastering and CDTV ISO build services: soon available
Terms of business: 90 days net

Disc Manufacturing, Inc.
1120 Cosby Way
Anaheim, CA 92086
tel. (1) 714-630-6700
fax (1) 714-630-1025
Sales contact: Wan Seegmiller
Technical contact: Leon Whidbee

You may also contact:
Disc Manufacturing, Inc.
A Quixote Company
4905 Moores Mill Rd.
Huntsville, AL. 35810
tel. (1) 205-859-9042
fax (1) 205-859-9932
Sales contact: Kim Vandenberghe
Technical contact: Shogo Karitani

Territories served: North and South America, Europe, Far East
Preferred Source format: 8mm Exabyte, Pinnacle MO, One-off CD, 9- track computer tape
One-off service: available
Pre-mastering and CDTV ISO build services: soon available
Terms of Business: Net 30 days, on credit approval

Next Technology Corporation Ltd.
St. John's Innovation Center
Cambridge
Cambridgeshire CB4 4WS
UNITED KINGDOM
tel (44) 223 420 222
fax (44) 223 420 015
Sales contact: Ian Thomas
Technical contact: Neil Critchell

Territories served: Europe
Pre-mastering and CDTV ISO building service: available

NEXT does not do replication. They are specialized in pre-mastering and CDTV ISO building services.

Nimbus Information Systems
SR 629, Guildford Farm
Ruckersville, VA 22968
USA
tel. (1) 804-985-1100 or 800-782-0778
fax (1) 804-985-4625
Sales contact: Larry Boden
Technical contact: Ernest Runyon

In Europe, contact:
Nimbus Information Systems
Wyastone Leys, Monmouth
GWENT NP5 3SR
United Kingdom
tel. (44) 600-890682
fax (44) 600-890779
Sales contact: Steve Connoly
Technical Contact: Jim Orr

Territories served: US and Europe
Preferred source format:4mm DAT or 8mm Exabyte
Terms of business: Net 30 days, upon credit approval
One-off service:available
Pre-mastering and CDTV ISO building service:available

Phillips And Du Pont Optical
1409 Foulk Road, Suite 200
Wilmington, DE 19803
tel. (1) 302-479-2501
fax (1) 302-479-2512
Sales contact: US: Joe Bradley tel. 301-989-9341
Technical contact: Jim Fricks tel. 704-734-4211

In Europe, contact:
Phillips And Du Pont Optical Co.
Buizerdlann 2,3435 SB
Nieuwegein, The Netherlands
tel. (31) 34-2-78754
Sales contact: Robert Fisher tel. (31) 3402-78754
Technical contact: Uwe Spitzenberger tel. (49) 511-7306253

Territories served: US and Europe
Preferred source format: 8mm, 9-track tape, one-off CD

Terms of business: Net 30 days
One-off service: available
Pre-mastering and CDTV ISO building service: soon available

Sonopress Gmbh
Carl-Bertelsmann-Str. 161
D-4830 Gutersloh 100
Germany
tel. (49) 5241-803074
fax (49) 5241-73686
Sales contact: Dr. Reinhard Raubenheimer
Technical contact: Mr. Ulrich Graznow tel. (49) 5241-805250
fax (49) 5241-75863
Territories served: All European countries, US if required
Preferred source format: 9 track tape, Exabyte, SCSI harddisk. Audio input media: Sony 1610/1630
Terms of business: based on customers requirements
One-off service: available
Pre-mastering service: soon available

Sonopress provides pan-European services. Here are contact names and numbers in other countries:

Sonopress UK
Sales contact: Miss Sabine Leuerer
tel. (44) 71-499-6813
fax (44) 71-493-7244

Sonopress France
Sales contact: Madame Bornhold
tel. (33) 1-45-636707
fax (33) 1-43-596673

Sonopress Italy
Sales contact: Dr. Paolo Montagna
tel. (39) 2-7600-4737
fax (39) 2-7601-5026

Appendix B

Sample Track Sheet

TITLE: SPACE WASTE   DEVELOPER: FLY BY NIGHT MULTIMEDIA

CD TRACK #	ROM/ TRACK NAME	GAP AUDIO	CD TIME	TIME START
01	AWESOME GAME	CDROM STD.	2 SEC	
02	SOUND EFFECTS TRACK	AUDIO	0 SEC.	04:15:00
03	INNA GODDA DAVIDA	AUDIO	0 SEC.	NA
04	FLY ME TO THE MOON	AUDIO	0 SEC.	NA

TOTAL TRACKS: 4

Notes:

Even though it seems obvious always be sure to write down the title and customer name at the top of the Track Sheet.

In this sample Track Sheet, track 1 contains data per the Yellow Book CD-ROM standards. Also notice that the 2 second post-gap is required (also per the Yellow Book).

Track 2 contains sound effects that will be played back via absolute Minute/Second/Frame reference (using PlayMSF). The track will begin at exactly 4 minutes 15 seconds CD time. Gap time (except for the mandatory CD-ROM gap) is not relevant since the program is addressing the track starting at an absolute location.

Track 3 is a soundtrack that is played during the opening sequences by playing the track (using PlayTrack). Absolute location is not needed since the PlayTrack function will start playing at the beginning of the track and will stop when directed by the program. Notice that the gap time is zero. The PlayTrack function will start playing at the beginning of the gap, not the beginning of the music. By using a gap time of zero, the music will start playing immediately instead of after the gap.

Track 4 is another soundtrack that is played exactly like track 3.

Be sure to write down the total number of tracks on the bottom of each Track Sheet that you submit. You wouldn't want a replicator to miss a track.

Simple Overview of Pre-Mastering Procedure

For a complete description of the procedure, please refer to the document ''Premastering and Mastering for CDTV'' in the CDTV Developers notes, April 8th edition. Here follows a simple outline:

1: BOOT with this disk

2: Open Shell

3: Copy the files ''rmtm'' and ''Bookit'' from the c: directory of this diskette to the c: directory of your application disk. Then copy the CDTV.TM file into the ROOT directory of your application disk.

4: Modify the startup-sequence of your application disk. Add the command rmtm at the beginning of the file. (If you include the Setpatch command, put rmtm just after setpatch.) You must also add the command Bookit in your startup-sequence. This command reads the Prefs in the CDTV memory, and centers the screen accordingly. Bookit has a number of switches. Type Bookit ? from the CLI for more information, or read the Bookit.doc file on this diskette.

5: type cd RAM:ISO2 <RTN>

6: type iso [options] <source path> <RTN>

```
Volume ISO9660:<Destination path>    <RTN>
VolumeIdentifier:(name i.e. Workbench) <RTN>
VolumeSetIdentifier:(name i.e. Volume1)   <RTN>
PublisherIdentifier:(your company name)   <RTN>
```

This commands creates a file named ''controlfile'', in the current directory. Controlfile is used by the buildtrack utility (see below).

[options] include the following:
    -n : sort by filename
    -s : sort by filesize
    default : no sort

The options define how the controlfile sorts the files in a directory: no sort, by filename, or by filesize. This lets you specify in which order you want your files laid out on the ISO disk image. A few examples:

ex) execute iso -n
ex) execute iso -s
ex) execute iso dh0:s
ex) execute iso -n dh0:utilities
ex) execute iso -s dh0:

The <source path> is the source disk or directory that will be built. This may be a full drive or just a single directory.

The <Destination path> indicates where the ISO 9660 image will be created.

If you are connected directly to the CD Mastering equipment, the Destinatioin path is normally DR1: In that case, you should skip steps 7 and 8. Alternatively, the Destination path can be any AmigaDOS filename (i.e. DH0:myfile). In this case, the file may be transferred to the CD Mastering equipment via a network, or serial or parallel lines.

The VolumeIdentifier, VolumeSetIdentifier, and PublisherIdentifier fields may contain any text string, containing no blanks.

After ISO controlfile is completed,

7: type bytedrive scsi.device 5 <RTN> (targetname unit#)

8: type mount dr1:<RTN>

9: type buildtrack controlfile -w -b 200 <RTN>

The -w option suppresses warnings concerning ISO file naming conventions.

The -b option specifies the number of buffers to use. 200 is the suggested value; you may increase or decrease this value according to the memory you have available. Default value is 64.

Source will then be copied to <Destination drive> in ISO 9660 format.

10: type fixtm <RTN>

This program looks for an ISO image on SCSI device 5. If it does not find one there, it will look for one on SCSI device 0.

a) If it finds one, it says: ''Device <n> contains an ISO image, volume name <yourvolume>. Update this image? (y/n)''

Type y <RTN>

b) If it doesn't find an ISO image on device 5 or 0, (you are building an ISO image on an AmigaDOS device) the program says: ''Input the name of AmigaDOS image file to be updated: ''. Here you should give the same filename you provided as a destination path in step 6 above, when prompted for ''Volume_ISO9660''.

This will complete the required process for the CDTV.TM file.

CAUTION !!!!!

ISO should be run in RAM DISK, running it on the source drive would, of course, modify the disk you are building as it is being built.

◆

CDTV and Meridian Data Products

Copyright Matthew Seitz, 1991. All rights reserved.

Second Edition, 8/1/91

Note: This paper assumes you are familiar with CD-ROM, ISO-9660, and Meridian Data products.

What is CDTV?

CDTV is Commodore, Inc.'s competitor to Philip's and Sony's CD-I product. Like CD-I, it is an interactive consumer electronic appliance based on compact discs. Like CD-I, CDTV not only specifies the physical format of the disc and the disc volume structure, but also the hardware and operating system of the playback system.

What is a CDTV Disc?

A CDTV disc is a CD-ROM disc with a Mode 1 data track. It may also have up to 98 audio tracks. Therefore CDTV discs are compatible with any standard Mode 1 compatible CD-ROM drive.

CDTV volume structure is a superset of Implementation Level 1, Interchange Level 2 ISO-9660 format. The main difference between ISO-9660 format and CDTV format is that CDTV allows File Identifiers to have lowercase letters and multiple "dot" (".", ASCII code 2E) characters. Because of these non-ISO-9660 File Identifiers, CDTV discs may or may not be readable by ISO-9660 compatible software.

How Do You Create a CDTV Pre-Mastering Tape or One-Off Disc Using Meridian Products?

There are two ways to do this:

1) Since a CDTV disc is a superset of a standard ISO-9660 CD-ROM, you can copy the CDTV files from an MS-DOS volume to an ISO-9660 formatted UDFAM partition using COPYISO. This assumes that you have the means to store your CDTV files as files on an MS-DOS formatted volume. You can then create a pre-master tape or one-off disc the same way you would with any ISO-9660 CD-ROM.

2) Since a CDTV disc is a standard Mode 1 CD-ROM, you can create an image file of your CDTV disc. By using the CD Publishing utility "DI2FILE", an image file can be created from any SCSI drive that contains an image of the CDTV disc. There are also CDTV authoring systems that can store an image file on tape. You can then transfer the image file to a pre-master tape or a one-off as you would with any CD-ROM image file.

What Else Do I Need to Produce CDTV Discs?

Nothing more is absolutely required to produce a CDTV disc. However, CDTV is intended to as a multimedia system. To efficiently produce CDTV files, you will want a CDTV authoring system, similar in concept to our CD Producer system. However, the CD Producer is NOT a CDTV authoring system. It does NOT produce CDTV compatible files. Meridian does not currently produce a CDTV authoring system. Customers who are interested in CDTV authoring systems should contact Commodore.

Which Facilities Master CDTV Discs?

To a mastering facility, CD-ROM should be CD-ROM. Mastering a CDTV disc should be no different than mastering any other ISO-9660 CD-ROM. For a current list of what facilities are actively supporting CDTV, contact Commodore.

How to Keep Your Sanity While Mastering Your CD

1) Always call for device CD0: in your code and never hard-code your devices.

2) Double-check and then triple-check your startup-sequence. If you can, test your startup-sequence on a floppy before you start. 80% of all pre-mastering failures are caused by a bad line or command in the startup-sequence.

3) Never assume that the CD-ROM will have the same volume name as your source disk. If you really, really need a particular volume name, them be sure to tell the pre-mastering technician. And tell him very clearly that you need this volume name.

4) Add the Workbench to your CD-ROM. You'll never know when you'll need it and you'll never miss 800K out of a 680 meg CD-ROM.

5) Set your screen colors to black but not until your disc is Gamma or Final. If your program drops into the CLI, then you can see it.

5) Watch your disk budget. Although CD-ROM's hold a phenomenal amount of data, there is a finite limit to their capacity. Believe it or not, there are developers who have run out of room for their titles. Remember these numbers: most one-off CD's hold less than 600mb per disc; the biggest manufactured disc holds about 680meg; CD Audio will take up just over 9meg per minute of audio time; ISO-9660 formatting will consume about 7meg of disc space; and there are mandatory gaps between your data and audio tracks that will take up disc space. Keep a close eye on the disc budget, it gets used up faster than you think.

6) Be sure you have removed all of your obsolete and embarrassing files. Don't make the same mistake that a certain company made with their demo disc.

7) Allow enough time for the pre-mastering techs to do the job. Don't call them every hour, or even every day. They are scrutinizing your disc to see that it will work on a CD-ROM. Every minute they spend on the phone with you is time they spend away from pre-mastering your disc. *Leave them alone.* The time required to pre-master depends entirely on the size of the disc, and the complexity of the startup-sequence. Action games that have a few files that fit in 5meg and a simple startup-sequence might be done in an hour or so. Gigantic discs with hundreds of files that need 600meg could take DAYS to pre-master..

8) Always submit complete file sets for pre-mastering. If you have a problem with one file, then don't ask the pre-mastering techs to just "add one more file" or "just change that file to say...". Rememeber that it's a stranger who is changing data on your disc.

9) Before you get to Beta, contact a few CD-ROM replicators and see what they can accept as an input medium. Some of our approved replicators can pre-master directly from an Amiga hard disk or floppy backups. . A few can't. Don't get caught two weeks from shipping final code with a hard disc your replicator can't turn into CD's. *Ask them before you reach beta.*

∞

Introduction to Locale.library

by Martin Taillefer

Localization is the process by which software dynamically adapts to different locales. A locale is made up of a set of attributes describing a language, a set of cultural and lingual conventions, a physical location, etc.

Without standardized system support to help deal with localization issues, the task of localizing applications is significant. There needs to be several different versions of every application, each specially adapted to run in a particular language and country.

Given the importance of the international market to the Amiga, it is imperative that the operating system provide services to facilitate, and thus encourage, application software localization. This is where *locale.library* comes in.

locale.library is an Amiga shared library offering services to let applications transparently adapt to any locale the user has chosen. Functions are provided for formatted information display, text catalog management, character attribute acquisition, string sorting, and more.

Compatibility-Now and in the Future

In a world where inter-platform compatibility is increasingly important, it is a significant advantage to adhere as closely as possible to adopted standards. This makes developing and porting applications to the Amiga easier, thus encouraging it.

locale.library offers the necessary services to implement fully compatible X/Open - ANSI/C localization support. Simple wrapper functions need to be provided by compiler writers to adhere to these standards, but the nuts and bolts of localization is in the library.

locale.library provides for easy and virtually unlimited expansion. All character manipulation routines are defined as accepting 32-bit characters, allowing future support for multi-byte character sets such as UniCode. Provisions are also made to allow the use of different character sets such as Cyrillic.

Since all language-specific text manipulation is implemented within language drivers, the library does not impose any artificial limitation on the simplicity or complexity of the algorithms used within the drivers. This enables the creation of very specialized sorting algorithms that adhere closely to a language's grammatical rules.

Workbench Disk

System localization is accomplished by an update to the Workbench disk. The changes needed include:

locale.library - This is the central element to localization and contains routines to allow management of locales, catalogs, and strings.

Language drivers - Much of *locale.library*'s functionality is implemented through language drivers. There is one such driver for every language supported. All the language drivers are stored in the LOCALE:Languages directory.

Country database - A set of files describing various attributes of every country supported, including the country's currency symbol, number format, measuring system, etc. Country files are stored in the LOCALE:Countries directory.

Locale preferences editor - A preferences editor found in the Prefs drawer that lets the user select which languages he speaks, the country in which he lives, and his time zone.

Revised system programs - Most programs found on the Workbench and Extras disks need to be updated in order to use *locale.library*.

System catalogs - A set of system text catalogs stored in the LOCALE:Catalogs directory. These provide all the strings needed to operate the system in different languages.

Application catalogs - A set of application text catalogs stored in either the PROGDIR:Catalogs or LOCALE:Catalogs directories. These are provided by applications and let them run in different languages.

Patching the ROM

When an updated version of IPrefs is run, it instructs *locale.library* to patch several ROM routines to transparently provide them with localized behavior.

The ROM routines patched by *locale.library* are:

exec.library/RawDoFmt() - Adds argument ordering support using nn$ specifications (see below for more information on this). Adds %U and %D formatting commands for localized number output.

dos.library/DosGetString() (internal routine) - Adds the ability to make use of disk-based text catalogs instead of always returning ROM-based strings.

dos.library/DateToStr() & dos.library/StrToDate() - Process and output localized dates

utility.library/ToLower() & utility.library/ToUpper() - Use the current language driver to adapt their behavior based on the user's current locale

utility.library/Strnicmp() & utility.library/Stricmp() - Use the current language driver to adapt their behavior based on the user's current locale

In addition, a new version of the LoadWB command patches a routine in workbench.library which enables the Workbench to run in the user's preferred language.

Disk Files

locale.library loads three types of files from disk to provide its functionality:

Preferences files - The user's locale preferences are stored as a standard prefs file named *ENV:Sys/locale.prefs*. The Locale prefs editor is provided to let the user control the contents of this file. An in-memory Locale structure is built directly based on a locale preferences file.

Language drivers - Each locale has a single language driver bound to it. When a locale is loaded in memory, a language driver is also loaded and automatically bound to the locale. The language drivers are located in LOCALE:Languages. For example:

LOCALE:Languages/dansk.language
LOCALE:Languages/francais.language
LOCALE:Languages/italiano.language

A language driver is an Amiga shared library providing around a dozen functions for string and character-oriented operations that need to be adapted depending on the language being used. The way each function is implemented is hidden to *locale.library*, allowing maximum flexibility for the driver when dealing with complex languages.

Message catalogs - Catalogs contain a series of translated strings for use by an application. They are stored on disk in an IFF file that is processed and managed by calls in the library. A utility called CatComp is provided that lets catalog files be generated easily.

Each locale contains a list of preferred languages. When a message catalog is accessed via the OpenCatalog() function, *locale.library* attempts to find a catalog in one of the locale's preferred languages. OpenCatalog() looks in two different places for message catalogs:

PROGDIR:Catalogs/languageName/catalogName.catalog
LOCALE:Catalogs/languageName/catalogName.catalog

languageName is one of the locale's preferred languages. catalogName is the name of the catalog the application wishes to open. Typically, catalogName is the name of the application.

Looking in the LOCALE: directory for catalogs lets the user put message catalogs for small applications (system tools for example) into a single directory, or even simpler on a single disk called LOCALE. In addition, using LOCALE: gives applications loaded as resident the ability to load in catalogs. Since resident applications do not get the benefit of PROGDIR:, LOCALE: becomes the automatic fall-back.

Of course, language drivers and catalogs are transparently cached by *locale.library* and get flushed whenever a memory panic occurs and when their respective use count is 0.

Functions

locale.library implements the actual localization facilities for application use. Around two dozen functions are available to access and manipulate locales, catalogs, and strings. Language drivers implement much of the functionality found in *locale.library*. The drivers offer services needed by the higher-level functions in the library. Each locale is bound automatically to a single language driver.

Locales

The two most basic routines in the library are OpenLocale() and CloseLocale(). OpenLocale() returns a pointer to a Locale structure, which can then be used with most of the other functions in the library. A Locale structure describes all the attributes necessary to localize an application and is built from the values found in a locale preferences file.

During the OpenLocale() call, a language driver is automatically bound to a locale. The language driver bound to the locale depends on the list of preferred languages specified by the user. There is one driver per supported language. Drivers are easy to write and new ones can be added by C= or third parties to support new languages as demand arises.

The language drivers offer specific services needed by the library to implement its functionality. This currently involves only text manipulation functions specifically adapted to the language at hand. For example, sorting text behaves differently from language to language and needs to be taken care of by specialized code.

Catalogs

Message catalogs are a simple means for an application to separate the strings of a program from the actual executable. By adding new message catalogs, the application can be made to support new languages and to display all of its output in the desired language.

Message catalogs are loaded in memory via the OpenCatalog() function. Two of the parameters required by this function are a locale and a catalog name. The locale is provided since it contains the list of the user's preferred languages. Based on these languages, OpenCatalog() attempts to find the best catalog possible from the series of catalogs provided by the application.

Once a message catalog is in memory, any string it contains can be retrieved using the GetCatalogStr() routine. The routine expects a number specifying which string of the catalog is to be returned. Sparse numbering of the strings is supported providing the most convenience possible to the application writer.

Date And Time

A Locale structure specifies special date and time formatting strings. These strings describe exactly how the date and time are to be displayed and interpreted. The FormatDate() routine is provided to convert an AmigaDOS DateStamp structure into a localized date string, while the ParseDate() routine converts a string to a DateStamp structure.

Argument Positioning

The grammar of most languages requires that subjects, objects, adverbs and complements be written out in a specific order. This order varies from language to language. For example:

''Please insert volume Workbench in drive DF0:''

might be said as:

''Insert in drive DF0: volume Workbench please''

To address this problem, an extension to the standard C-language printf() conventions used by FormatString() is argument position specification. Specifying the argument position lets the order of the % commands change while the arguments provided remain the same. Using the C printf() call as an example:

```
eyes = 2;
feet = 3;
ears = 4;
printf("%d eyes, %d feet and %d ears",eyes,feet,ears);
printf("%3$d ears, %1$d eyes and %2$d feet",eyes,feet,ears);
```

These two statements would produce the following output:

''2 eyes, 3 feet and 4 ears'' for the first
''4 ears, 2 eyes and 3 feet'' for the second

The argument positioning feature lets you change the format string being processed while keeping the data stream the same. This is an invaluable tool when translating strings to different languages.

So, in practical terms, you can change the order in which things appear. As an example, the following string from the workbench catalog:

''Amiga Workbench %lD graphics mem %lD other mem''

can be changed to:

''Amiga Workbench %2$lD other mem %1$lD graphics mem''

and the string produced and displayed in the title bar of the Workbench would be correct.

locale.library patches *exec.library*/RawDoFmt() to give it support for argument ordering. In addition, locale's own FormatString() routine supports argument ordering as well.

Sorting

Sorting follows different rules based on the language of the strings being sorted. Thus, *locale.library* provides two routines to deal with sorting: StrnCmp() and StrConvert().

StrnCmp() is similar to the C function of the same name, except that it takes an argument specifying which type of sort to perform. There are currently three supported sorts, each having its own role. Refer to the autodoc entry for the StrnCmp() function for more information.

StrConvert() is a routine to enhance performance. Its purpose in life is to convert a string into a special format that can be compared efficiently. That is, the more advanced comparison types offered by StrnCmp() can be several times slower than a standard ASCII-based strncmp() call in C because StrnCmp() takes into account many more factors when sorting.

In programs, it is often necessary to compare a single string with many different strings. Think for example of how alphabetical insertion into a list is done. In these cases, the slower operation of StrnCmp() vs strncmp() can impact performance. To alleviate this, StrConvert() allows you to perform most of the work performed by StrnCmp() only once. Once a string has been converted by StrConvert(), it can be used with regular C strncmp() calls.

To clarify, given two strings A and B, the following will produce equivalent results:

```
StrnCmp(A,B)  ==  StrConvert(A,a)
StrConvert(B,b)
strncmp(a,b)
```

Example

Here is a sample localized C program. This program requires a message catalog called sample.catalog to get its strings from.

```
/* localetest.c */
#include <exec/types.h>
#include <exec/libraries.h>
#include <libraries/locale.h>

#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/locale_protos.h>


/*****************************************************************************/


extern struct Library *DOSBase;
extern struct Library *SysBase;
       struct Library *LocaleBase;
       struct Catalog *catalog = NULL;


/*****************************************************************************/


enum AppStringsID
{
    MSG_HELLO,
    MSG_BYE
};


/*****************************************************************************/
```

```
STRPTR AppStrings[] =
{
    "Hello World!\n",
    "Bye, I'm leaving...\n"
};

/*****************************************************************************/


STRPTR GetString(enum AppStringsID id)
{
    if (LocaleBase)
        return(GetCatalogStr(catalog,id,AppStrings[id]));

    return(AppStrings[id]);
}

/*****************************************************************************/


VOID main(VOID)
{
    if (LocaleBase = OpenLibrary("locale.library",38))
        catalog = OpenCatalogA(NULL,"sample.catalog",NULL);

    PutStr(GetString(MSG_HELLO));
    PutStr(GetString(MSG_BYE));

    if (LocaleBase)
    {
        CloseCatalog(catalog);
        CloseLibrary(LocaleBase);
    }
}
```

Structures

The Locale structure is the main public structure provided by *locale.library*. The structure is defined in <libraries/locale.h> and consists of the following fields:

STRPTR loc_LocaleName - Locale's name.
STRPTR loc_LanguageName - The language of the driver bound to this locale.
STRPTR loc_PrefLanguages[10] - The ordered list of preferred languages for this locale.
ULONG loc_Flags - Locale flags. The single current flag specifies whether daylight savings time should be used in the system.
ULONG loc_CodeSet - Specifies the code set required by this locale. Currently, this value is always 0.
ULONG loc_CountryCode - The international country code.
ULONG loc_TelephoneCode - The international telephone code for the country.
LONG loc_GMTOffset - The offset in minutes of the current location from GMT.
UBYTE loc_MeasuringSystem - The measuring system being used.
STRPTR loc_DateTimeFormat - The date and time format string, ready to pass to FormatDate()
STRPTR loc_DateFormat - The date format string.
STRPTR loc_TimeFormat - The time format string.
STRPTR loc_ShortDateTimeFormat - The short date and time format string, ready to pass to FormatDate()
STRPTR loc_ShortDateFormat - The short date format string.
STRPTR loc_ShortTimeFormat - The short time format string.
STRPTR loc_DecimalPoint - The decimal point character used to format non-monetary quantities.

STRPTR loc_GroupSeparator - The characters used to separate groups of digits before the decimal-point character in formatted non-monetary quantities.

STRPTR loc_FracGroupSeparator - The characters used to separate groups of digits after the decimal-point character in formatted non-monetary quantities.

STRPTR loc_Grouping - A string whose elements indicate the size of each group of digits before the decimal-point character in formatted non-monetary quantities.

STRPTR loc_FracGrouping - A string whose elements indicate the size of each group of digits after the decimal-point character in formatted non-monetary quantities.

STRPTR loc_MonDecimalPoint - The decimal point used to format monetary quantities.

STRPTR loc_MonGroupSeparator - The separator for groups of digits before the decimal point in monetary quantities.

STRPTR loc_MonFracGroupSeparator - The separator for groups of digits after the decimal point in monetary quantities.

STRPTR loc_MonGrouping - A string whose elements indicate the size of each group of digits before the decimal-point character in monetary quantities.

STRPTR loc_MonFracGrouping - A string whose elements indicate the size of each group of digits after the decimal-point character in monetary quantities.

UBYTE loc_MonFracDigits - The number of fractional digits (those after the decimal-point) to be displayed in a formatted monetary quantity.

UBYTE loc_MonIntFracDigits - The number of fractional digits (those after the decimal-point) to be displayed in an internationally formatted monetary quantity.

STRPTR loc_MonCS - The local currency symbol applicable to the current locale.

STRPTR loc_MonSmallCS - The currency symbol for small amounts.

STRPTR loc_MonIntCS - The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in ISO 4217 Codes for the Representation of Currency and Funds. The fourth character (immediately preceding the NULL) is the character used to separate the international currency symbol from the monetary quantity.

STRPTR loc_MonPositiveSign - The string used to indicate a nonnegative-valued formatted monetary quantity.

UBYTE loc_MonPositiveSpaceSep - Specifies the number of spaces separating the currency symbol from the non-negative monetary quantity.

UBYTE loc_MonPositiveSignPos - Set to a value indicating the positioning of loc_MonPositiveSign for a non-negative monetary quantity.

UBYTE loc_MonPositiveCSPos - Set to 1 or 0 if loc_MonCS respectively precedes or succeeds the value for a non-negative monetary quantity.

STRPTR loc_MonNegativeSign - The string used to indicate a negative-valued monetary quantity.

UBYTE loc_MonNegativeSpaceSep - Specifies the number of spaces separating the currency symbol from the negative monetary quantity.

UBYTE loc_MonNegativeSignPos - Set to a value indicating the positioning of loc_MonNegativeSign for a negative formatted monetary quantity.

UBYTE loc_MonNegativeCSPos - Set to 1 or 0 if loc_MonCS respectively precedes or succeeds the value for a negative formatted monetary quantity.

The grouping tables pointed to by loc_Grouping, loc_FracGrounping, loc_MonGrouping, and loc_MonFracGrouping contain a stream of bytes with the following values:

255	No further grouping is to be performed.
0	The previous element is to be repeatedly used for the remainder of the digits.
1..254	The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

The values of loc_MonPositiveSignPos and loc_MonNegativeSignPos are interpreted according to the following:

0	Parentheses surround the quantity and currency symbol
1	The sign string precedes the quantity and currency symbol
2	The sign string succeeds the quantity and currency symbol
3	The sign string immediately precedes the currency symbol
4	The sign string immediately succeeds the currency symbol.

Using Locale.library from ARexx

locale.library provides an ARexx function host interface enabling ARexx programs to take advantage of system localization. The functions provided by the interface are directly analogous to the functions available to C and assembly programmers, with the differences mostly being in the way they are called.

The function host library vector is located at offset -30 from the library. This is the value you provide to ARexx in the AddLib() function call. Here is a sample ARexx script to add *locale.library* to ARexx's function host list:

```
/* Make sure the library is loaded as a function host */
IF ~SHOW(L,’locale.library’) THEN DO
  CALL ADDLIB(’locale.library’,0,-30)
END;
```

As mentioned, the functions available through the function host are very similar to the ones available from C and assembly. Here is a list of the commands supported through the function host. Each command has an AmigaDOS-style template that describes the arguments it accepts, along with a short description of what the command does. Refer to the autodoc entries with the same names for more complete explanations of what each command is intended for.

CloseCatalog (CATALOG/N/A) - Closes a previously opened message catalog. Pass it the return value from OpenCatalog().

ConvToLower (CHARACTER/A) - Pass it the character to convert and it returns the converted character.

ConvToUpper (CHARACTER/A) - Pass it the character to convert and it returns the converted character.

GetCatalogStr (CATALOG/A,STRING/N/A,DEFAULT/A) - Extract a string from a catalog. Pass it the catalog obtained from OpenCatalog(), the number of the string to extract from the catalog, and a default string to return in case the requested string is not in the catalog.

IsAlNum (CHARACTER/A) - Pass it a character and it returns 1 if the character is alphanumeric, or returns 0 otherwise

IsAlpha (CHARACTER/A) - Pass it a character and it returns 1 if the character is alphabetic, or returns 0 otherwise

IsCntrl (CHARACTER/A) - Pass it a character and it returns 1 if the character is a control character, or returns 0 otherwise

IsDigit (CHARACTER/A) - Pass it a character and it returns 1 if the character is a digit, or returns 0 otherwise

IsGraph (CHARACTER/A) - Pass it a character and it returns 1 if the character can be represented graphically, or returns 0 otherwise

IsLower (CHARACTER/A) - Pass it a character and it returns 1 if the character is lower case, or returns 0 otherwise

IsPrint (CHARACTER/A) - Pass it a character and it returns 1 if the character is a space, or returns 0 otherwise

IsPunct (CHARACTER/A) - Pass it a character and it returns 1 if the character is a punctuation mark, or returns 0 otherwise

IsSpace (CHARACTER/A) - Pass it a character and it returns 1 if the character is a white space, or returns 0 otherwise

IsUpper (CHARACTER/A) - Pass it a character and it returns 1 if the character is upper case, or returns 0 otherwise

IsXDigit (CHARACTER/A) - Pass it a character and it returns 1 if the character is a hexadecimal digit (0..9, a..f, A..F), or returns 0 otherwise

OpenCatalog (NAME/A,BUILTIN/A,VERSION/N/A) - Opens a message catalog and returns a value to be used for GetCatalogStr() and CloseCatalog(). The first parameter specifies the name of the catalog to load, the second parameter specifies the language used for the default strings in the script, and the final parameter specifies the version number of the catalog to open. A version of 0 means any version is fine.

StrnCmp (STRING1/A,STRING2/A,TYPE/N/A) - Compares two strings and returns -1 if STRING1 comes before STRING2, returns 0 if both strings are equal, and returns 1 if STRING1 comes after STRING2. The TYPE parameter is either 0 to do an ASCII comparison, 1 to do a single pass collation, or 2 to do a two pass collation (this corresponds to the C SC_ASCII, SC_COLLATE1 and SC_COLLATE2 constants explained in the autodoc entry for StrnCmp()).

Here is a sample ARexx script that exercises some of locale's function host commands:

```
/* localetest.rexx */

/* Make sure locale is loaded as a function host */
IF ~SHOW(L,'locale.library') THEN DO
  CALL ADDLIB('locale.library',0,-30);
END;

say ConvToLower("A");
say ConvToUpper("b");
say IsAlpha("1");

catalog = OpenCatalog("workbench.catalog","english",0);
say GetCatalogStr(catalog,34,"default");
say CloseCatalog(catalog);
say StrnCmp("abc","def",2);
```

Appendix A - Includes

```
#ifndef LIBRARIES_LOCALE_H
#define LIBRARIES_LOCALE_H
/*
**      $Id: locale.h,v 38.4 91/07/25 13:00:28 vertex Exp $
**
**      locale.library interface structures and definitions
**
**      (C) Copyright 1991 Commodore-Amiga, Inc.
**      All Rights Reserved
*/


#ifndef EXEC_TYPES_H
#include <exec/types.h>
#endif

#ifndef EXEC_NODES_H
#include <exec/nodes.h>
#endif

#ifndef EXEC_LISTS_H
#include <exec/lists.h>
#endif

#ifndef UTILITY_TAGITEM_H
#include <utility/tagitem.h>
#endif

/*****************************************************************************/


/* constants for GetLocaleStr() */
#define DAY_1           1       /* Sunday    */
#define DAY_2           2       /* Monday    */
#define DAY_3           3       /* Tuesday   */
#define DAY_4           4       /* Wednesday */
#define DAY_5           5       /* Thursday  */
#define DAY_6           6       /* Friday    */
#define DAY_7           7       /* Saturday  */

#define ABDAY_1         8       /* Sun */
#define ABDAY_2         9       /* Mon */
#define ABDAY_3         10      /* Tue */
#define ABDAY_4         11      /* Wed */
#define ABDAY_5         12      /* Thu */
#define ABDAY_6         13      /* Fri */
#define ABDAY_7         14      /* Sat */

#define MON_1           15      /* January   */
#define MON_2           16      /* February  */
#define MON_3           17      /* March     */
#define MON_4           18      /* April     */
#define MON_5           19      /* May       */
#define MON_6           20      /* June      */
#define MON_7           21      /* July      */
#define MON_8           22      /* August    */
#define MON_9           23      /* September */
#define MON_10          24      /* October   */
#define MON_11          25      /* November  */
#define MON_12          26      /* December  */

#define ABMON_1         27      /* Jan    */
#define ABMON_2         28      /* Feb    */
#define ABMON_3         29      /* Mar    */
#define ABMON_4         30      /* Apr    */
#define ABMON_5         31      /* May    */
#define ABMON_6         32      /* Jun    */
#define ABMON_7         33      /* Jul    */
#define ABMON_8         34      /* Aug    */
#define ABMON_9         35      /* Sep    */
#define ABMON_10        36      /* Oct    */
#define ABMON_11        37      /* Nov    */
#define ABMON_12        38      /* Dec    */
```

```
#define YESSTR          39      /* affirmative response for yes/no queries */
#define NOSTR           40      /* negative response for yes/no queries    */

#define AM_STR          41      /* AM                                      */
#define PM_STR          42      /* PM                                      */

#define SOFTHYPHEN      43      /* soft hyphenation                        */
#define HARDHYPHEN      44      /* hard hyphenation                        */

#define OPENQUOTE       45      /* start of quoted block                   */
#define CLOSEQUOTE      46      /* end of quoted block                     */

#define MAXSTRMSG       47      /* current number of defined strings       */

/*****************************************************************************/


/* This structure must only be allocated by locale.library and is READ-ONLY! */
struct Locale
{
    STRPTR      loc_LocaleName;             /* locale's name               */
    STRPTR      loc_LanguageName;           /* language of this locale     */
    STRPTR      loc_PrefLanguages[10];      /* preferred languages         */
    ULONG       loc_Flags;                  /* see below                   */

    ULONG       loc_CodeSet;                /* for now, always 0           */
    ULONG       loc_CountryCode;            /* user's country code         */
    ULONG       loc_TelephoneCode;          /* country's telephone code    */
    LONG        loc_GMTOffset;              /* minutes from GMT            */
    UBYTE       loc_MeasuringSystem;        /* what measuring system?      */
    UBYTE       loc_Reserved0[3];

    STRPTR      loc_DateTimeFormat;         /* regular date & time format  */
    STRPTR      loc_DateFormat;             /* date format by itself       */
    STRPTR      loc_TimeFormat;             /* time format by itself       */

    STRPTR      loc_ShortDateTimeFormat;    /* short date & time format    */
    STRPTR      loc_ShortDateFormat;        /* short date format by itself */
    STRPTR      loc_ShortTimeFormat;        /* short time format by itself */

    /* for numeric values */
    STRPTR      loc_DecimalPoint;           /* character before the decimals    */
    STRPTR      loc_GroupSeparator;         /* separates groups of digits       */
    STRPTR      loc_FracGroupSeparator;     /* separates groups of digits       */
    UBYTE      *loc_Grouping;               /* size of each group               */
    UBYTE      *loc_FracGrouping;           /* size of each group               */

    /* for monetary values */
    STRPTR      loc_MonDecimalPoint;
    STRPTR      loc_MonGroupSeparator;
    STRPTR      loc_MonFracGroupSeparator;
    UBYTE      *loc_MonGrouping;
    UBYTE      *loc_MonFracGrouping;
    UBYTE       loc_MonFracDigits;          /* digits after the decimal point  */
    UBYTE       loc_MonIntFracDigits;       /* for international representation*/
    UBYTE       loc_Reserved1[2];

    /* for currency symbols */
    STRPTR      loc_MonCS;                  /* currency symbol                  */
    STRPTR      loc_MonSmallCS;             /* symbol for small amounts         */
    STRPTR      loc_MonIntCS;               /* internationl (ISO 4217) code     */

    /* for positive monetary values */
    STRPTR      loc_MonPositiveSign;        /* indicate positive money value    */
    UBYTE       loc_MonPositiveSpaceSep;    /* determine if separated by space */
    UBYTE       loc_MonPositiveSignPos;     /* position of positive sign        */
    UBYTE       loc_MonPositiveCSPos;       /* position of currency symbol      */
    UBYTE       loc_Reserved2;

    /* for negative monetary values */
    STRPTR      loc_MonNegativeSign;        /* indicate negative money value    */
    UBYTE       loc_MonNegativeSpaceSep;    /* determine if separated by space */
    UBYTE       loc_MonNegativeSignPos;     /* position of negative sign        */
    UBYTE       loc_MonNegativeCSPos;       /* position of currency symbol      */
    UBYTE       loc_Reserved3;
};
```

```
/* flags for Locale.loc_Flags */
#define LOCB_USES_DST   0       /* user uses daylight savings time? */
#define LOCF_USES_DST   (1<<0)

/* constants for Locale.loc_MeasuringSystem */
#define MS_ISO          0       /* international metric system */
#define MS_AMERICAN     1       /* american system             */
#define MS_IMPERIAL     2       /* imperial system             */
#define MS_BRITISH      3       /* british system              */

/* constants for Locale.loc_MonPositiveSpaceSep and Locale.loc_MonNegativeSpaceSep */
#define SS_NOSPACE 0  /* cur. symbol is NOT separated from value with a space */
#define SS_SPACE   1  /* cur. symbol IS separated from value with a space     */

/* constants for Locale.loc_MonPositiveSignPos and Locale.loc_MonNegativeSignPos */
#define SP_PARENS    0  /* () surround the quantity and currency_symbol   */
#define SP_PREC_ALL  1  /* sign string comes before amount and symbol     */
#define SP_SUCC_ALL  2  /* sign string comes after amount and symbol      */
#define SP_PREC_CURR 3  /* sign string comes right before currency symbol */
#define SP_SUCC_CURR 4  /* sign string comes right after currency symbol  */

/* constants for Locale.loc_MonPositiveCSPos and Locale.loc_MonNegativeCSPos */
#define CSP_PRECEDES 0  /* currency symbol comes before value */
#define CSP_SUCCEEDS 1  /* currency symbol comes after value  */

/* elements of the byte arrays pointed to by:
 *   Locale.loc_Grouping
 *   Locale.loc_FracGrouping
 *   Locale.loc_MonGrouping
 *   Locale.loc_MonFracGrouping
 * are interpreted as follows:
 *
 *    255     indicates that no further grouping is to be performed
 *    0       indicates that the previous element is to be repeatedly used
 *            for the remainder of the digits
 *   <other> the number of digits that comprises the current group
 */

/******************************************************************************/


/* Tags for OpenCatalog() */
#define OC_TagBase         (TAG_USER + 0x90000)
#define OC_BuiltInLanguage OC_TagBase+1   /* language of built-in strings    */
#define OC_BuiltInCodeSet  OC_TagBase+2   /* code set of built-in strings    */
#define OC_Version         OC_TagBase+3   /* catalog version number required */
#define OC_Language        OC_TagBase+4   /* preferred language of catalog   */


/******************************************************************************/


/* Comparison types for StrnCmp() */
#define SC_ASCII    0
#define SC_COLLATE1 1
#define SC_COLLATE2 2


/******************************************************************************/


/* This structure must only be allocated by locale.library and is READ-ONLY! */
struct Catalog
{
    struct Node cat_Link;       /* for internal linkage    */
    UWORD       cat_Pad;        /* to longword align       */
    STRPTR      cat_Language;   /* language of the catalog */
    ULONG       cat_CodeSet;    /* currently always 0      */
    UWORD       cat_Version;    /* version of the catalog  */
    UWORD       cat_Revision;   /* revision of the catalog */
};


/******************************************************************************/


#endif  /* LIBRARIES_LOCALE_H */
```

Appendix B - Locale.library Autodoc

```
TABLE OF CONTENTS

locale.library/--rexxhost--
locale.library/CloseCatalog
locale.library/CloseLocale
locale.library/ConvToLower
locale.library/ConvToUpper
locale.library/FormatDate
locale.library/FormatString
locale.library/GetCatalogStr
locale.library/GetLocaleStr
locale.library/IsXXXX
locale.library/OpenCatalog
locale.library/OpenLocale
locale.library/ParseDate
locale.library/StrConvert
locale.library/StrnCmp




locale.library/--rexxhost--                         locale.library/--rexxhost--

   HOST INTERFACE
    locale.library provides an ARexx function host interface that enables
    ARexx programs to take advantage of system localization. The
    functions provided by the interface are directly related to the
    functions described herein, with the differences mostly being
    in the way they are called.

    The function host library vector is located at offset -30 from the
    library. This is the value you provide to ARexx in the AddLib()
    function call.

   FUNCTIONS
    CloseCatalog (CATALOG/N/A)
    ConvToLower (CHARACTER/A)
    ConvToUpper (CHARACTER/A)
    GetCatalogStr (CATALOG/A,STRING/N/A,DEFAULT/A)
    IsAlNum (CHARACTER/A)
    IsAlpha (CHARACTER/A)
    IsCntrl (CHARACTER/A)
    IsDigit (CHARACTER/A)
    IsGraph (CHARACTER/A)
    IsLower (CHARACTER/A)
    IsPrint (CHARACTER/A)
    IsPunct (CHARACTER/A)
    IsSpace (CHARACTER/A)
    IsUpper (CHARACTER/A)
    IsXDigit (CHARACTER/A)
    OpenCatalog (NAME/A,BUILTIN/A,VERSION/N/A)
    Strncmp (STRING1/A,STRING2/A,TYPE/N/A)
```

```
locale.library/CloseCatalog                     locale.library/CloseCatalog

   NAME
    CloseCatalog -- close a message catalog. (V38)

   SYNOPSIS
    CloseCatalog(catalog);
                 A0

    VOID CloseCatalog(struct Catalog *);

   FUNCTION
    Concludes access to a message catalog. The usage count of the
    catalog is decremented. When this count reaches 0, the catalog
    can be expunged from system memory whenever a memory panic occurs.

   INPUTS
    catalog - the message catalog to close. A NULL catalog is a valid
          parameter and is simply ignored.

   SEE ALSO
    OpenCatalog(), GetCatalogStr()
```

```
locale.library/CloseLocale                       locale.library/CloseLocale

   NAME
    CloseLocale -- close a locale. (V38)

   SYNOPSIS
    CloseLocale(locale);
                A0

    VOID CloseLocale(struct Locale *);

   FUNCTION
    Concludes access to a locale.

   INPUTS
    locale - an opened locale. A NULL locale is a valid
         parameter and is simply ignored.

   SEE ALSO
    OpenLocale(), <libraries/locale.h>
```

```
locale.library/ConvToLower                           locale.library/ConvToLower

   NAME
    ConvToLower -- convert a character to lower case. (V38)

   SYNOPSIS
    char = ConvToLower(locale,character);
    D0                 A0     D0

    ULONG ConvToLower(struct Locale *,ULONG);

   FUNCTION
    This function tests if the character specified is upper case. If it is
    then the lower case version of that character is returned, and if it
    isn't then the original character is returned.

   INPUTS
    locale - the locale to use for the conversion
    character - the character to convert

   RESULTS
    char - a (possibly) converted character

   NOTE
    This function requires a full 32-bit character be passed-in in order
    to support multi-byte character sets.
```

```
locale.library/ConvToUpper                           locale.library/ConvToUpper

   NAME
    ConvToUpper -- convert a character to upper case. (V38)

   SYNOPSIS
    char = ConvToUpper(locale,character);
    D0                 A0     D0

    ULONG ConvToUpper(struct Locale *,ULONG);

   FUNCTION
    This function tests if the character specified is lower case. If it is
    then the upper case version of that character is returned, and if it
    isn't then the original character is returned.

   INPUTS
    locale - the locale to use for the conversion
    character - the character to convert

   RESULTS
    char - a (possibly) converted character

   NOTE
    This function requires a full 32-bit character be passed-in in order
    to support multi-byte character sets.
```

```
locale.library/FormatDate                              locale.library/FormatDate

   NAME
    FormatDate -- generate a date string based on a date formatting
              template. (V38)

   SYNOPSIS
    FormatDate(locale,string,date,putCharFunc);
               A0     A1     A2   A3

    VOID FormatDate(struct Locale *,STRPTR,struct DateStamp *,
                    struct Hook *);

   FUNCTION
    This function processes a formatting template and generates
    a stream of bytes that's sent one character at a time to the
    putCharFunc callback hook.

   INPUTS
    locale - the locale to use for the formatting
    string - the NULL-terminated template describing the desired format
         for the date. This is constructed just like C-language
         printf() statements, except that different formatting codes
             are used. Just like in C, formatting codes start with a
             % followed by the formatting command. The following
         commands are accepted by this function:

                 %a - abbreviated weekday name
                 %A - weekday name
                 %b - abbreviated month name
                 %B - month name
                 %c - same as "%a %b %d %H:%M:%S %Y"
                 %C - same as "%a %b %e %T %Z %Y"
                 %d - day number with leading 0s
                 %D - same as "%m/%d/%y"
                 %e - day number with leading spaces
                 %h - abbreviated month name
                 %H - hour using 24-hour style
                 %I - hour using 12-hour style
                 %j - julian date
                 %m - month number with leading 0s
                 %M - the number of minutes with leading 0s
                 %n - insert a linefeed
                 %p - AM or PM strings
                 %r - same as "%I:%M:%S %p"
                 %R - same as "%H:%M"
                 %S - number of seconds with leadings 0s
                 %t - insert a tab character
                 %T - same as "%H:%M:%S"
                 %U - week number, taking Sunday as first day of week
                 %w - weekday number
                 %W - week number, taking Monday as first day of week
                 %x - same as "%m/%d/%y"
                 %X - same as "%H:%M:%S"
                 %y - year using two digits with leading 0s
                 %Y - year using four digits with leading 0s

             If the template parameter is NULL, a single NULL byte
         is sent to putCharFunc.
    date - the date to format into a string
    putCharFunc - a callback hook invoked for every character generated,
              including for the terminating NULL character. The hook
              is called with:

            A0 - address of Hook structure
            A1 - character for hook to process (not a pointer!)
                    A2 - locale pointer

   SEE ALSO
    ParseDate(), <libraries/locale.h>, <dos/dos.h>
```

```
locale.library/FormatString                          locale.library/FormatString

   NAME
    FormatString -- format data into a character stream. (V38)

   SYNOPSIS
    next = FormatString(locale,string,dataStream,putCharFunc);
    D0                  A0     A1     A2         A3

    APTR FormatString(struct LocaleBase *,STRPTR,APTR,struct Hook *);

   FUNCTION
    This function performs C-language-like formatting of a data stream,
    outputting the result a character at a time. Where % formatting
    commands are found in the string, they are replaced with the
    corresponding elements in 'dataStream'. %% must be used in the
    string if a % is desired in the output.

    An extension to the standard C-language printf() conventions used
    by FormatString() is argument position specification. Specifying the
    argument position lets the order of the % commands change while the
    arguments provided remain the same. Using the C printf() call as an
    example:
        printf("%d eyes, %d feet and %d ears",eyes,feet,ears);
        printf("%3$d ears, %1$d eyes and %2$d feet",eyes,feet,ears);
    These two statements would produce the following output:
        "2 eyes, 3 feet and 4 ears" for the first
          "4 ears, 2 eyes and 3 feet" for the second

    The argument positioning feature lets you change the format string
    being processed while keeping the data stream the same. This is
    an invaluable tool when translating strings to different languages.

   INPUTS
    locale - the locale to use for the formatting
    string - a C-language-like NULL-terminated format string,
         with the following supported % options:

      %[arg_pos$][flags][width][.limit][length]type

      arg_pos - ordinal position of the argument for this command within
            the array of arguments pointed to by 'dataStream'
        $      - must follow the arg_pos value, if specified
      flags    - only one allowed. '-' specifies left justification.
      width    - field width. If the first character is a '0', the
                 field is padded with leading 0s.
        .      - must precede the field width value, if specified
      limit    - maximum number of characters to output from a string.
                 (only valid for %s or %b).
      length   - size of input data defaults to word (16-bit) for types c,
            d, u and x, 'l' changes this to long (32-bit).
      type     - supported types are:
                       b - BSTR, data is 32-bit BPTR to byte count followed
                           by a byte string. A NULL BPTR is treated as an
                   empty string.
                       d - signed decimal
               D - signed decimal using the locale's formatting
                   conventions
               u - unsigned decimal
               U - unsigned decimal using the locale's formatting
                   conventions
                       x - hexadecimal
                       s - string, a 32-bit pointer to a NULL-terminated
                           byte string. A NULL pointer is treated
                           as an empty string.
                       c - character

              If the string parameter is NULL, the function returns without
         outputting anything.
    dataStream - a stream of data that is interpreted according to
              the format string. Often this is a pointer into
              the task's stack.
    putCharFunc - a callback hook invoked for every character generated,
              including for the terminating NULL character. The hook
              is called with:

            A0 - address of Hook structure
         A1 - character for hook to process (not a pointer!)
                 A2 - locale pointer

          the function is called with a NULL char at the end of
          the format string.

 RESULTS
  next - A pointer to beyond the last data element used in 'dataStream'
         (the next argument that would have been processed).
         This allows multiple formatting passes to be made using the
         same data.

 WARNING
  This function formats word values in the data stream. If your compiler
  defaults to longs, you must add an "l" to your specifications. This
  can get strange for characters, which might look like "%lc".

 SEE ALSO
  exec.library/RawDoFmt()
```

```
locale.library/GetCatalogStr                     locale.library/GetCatalogStr

   NAME
    GetCatalogStr -- get a string from a message catalog. (V38)

   SYNOPSIS
    string = GetCatalogStr(catalog,stringNum,defaultString);
    D0                      A0      D0        A1

    STRPTR GetCatalogStr(struct Catalog *,LONG,STRPTR);

   FUNCTION
    This function returns a specific string within a message catalog.
    If the catalog parameter is NULL, or the requested message does not
    exist, then defaultString is returned.

   INPUTS
    catalog - a message catalog as obtained from OpenCatalog(), or NULL
    stringNum - a message number within the catalog
    defaultString - string to return in case "stringNum" can't be found

   RESULTS
    string - a pointer to a NULL-terminated string. The returned string
         is READ-ONLY, do NOT modify! This string pointer is valid
         only as long as the catalog remains open.

   SEE ALSO
    OpenCatalog(), CloseCatalog()
```

```
locale.library/GetLocaleStr                     locale.library/GetLocaleStr

   NAME
    GetLocaleStr -- get a standard string from a locale. (V38)

   SYNOPSIS
    string = GetLocaleStr(locale,stringNum);
    D0                    A0     D0

    STRPTR GetLocaleStr(struct Locale *,ULONG);

   FUNCTION
    This function returns a specific string associated with the given
    locale.

   INPUTS
    locale - a valid locale
    stringNum - the number of the string to get a pointer to. See the
            constants defined in <libraries/locale.h> for the
            possible values.

   RESULTS
    string - a pointer to a NULL-terminated string, or NULL if the
         requested string number was out of bounds. The returned
         string is READ-ONLY, do NOT modify! This string pointer
         is valid only as long as the locale remains open.

   SEE ALSO
    OpenLocale(), CloseLocale(), <libraries/locale.h>
```

```
locale.library/IsXXXX                                   locale.library/IsXXXX

   NAME
    IsXXXX -- determine whether a character is of a certain type. (V38)

   SYNOPSIS
    state = IsXXXX(locale,character);
    D0             A0     D0

    BOOL IsXXXX(struct Locale *,ULONG);

   FUNCTION
    These functions determine whether the character specified is of a
    certain type, according to the supplied locale.

    IsAlNum() - test if alphanumeric character
    IsAlpha() - test if alphabetical character
    IsCntrl() - test if control character
    IsDigit() - test if decimal digit character
    IsGraph() - test if visible character
    IsLower() - test if lower case character
    IsPrint() - test if blank
    IsPunct() - test if punctuation character
    IsSpace() - test if white space character
    IsUpper() - test if upper case character
    IsXDigit() - test if hexadecimal digit

   INPUTS
    locale - the locale to use for the test
    character - the character to test

   RESULTS
    state - TRUE if the character is of the required type, FALSE otherwise

   NOTE
    These functions require full 32-bit characters be passed-in in order
    to support multi-byte character sets.
```

```
locale.library/OpenCatalog                          locale.library/OpenCatalog

   NAME
    OpenCatalogA -- open a message catalog. (V38)
    OpenCatalog -- varargs stub for OpenCatalogA(). (V38)

   SYNOPSIS
    catalog = OpenCatalogA(locale,name,tagList);
    D0                     A0     A1   A2

    struct Catalog *OpenCatalogA(struct Locale *,STRPTR,struct TagItem *);

    catalog = OpenCatalog(locale,name,firstTag, ...);

    struct Catalog *OpenCatalog(struct Locale *,STRPTR,Tag, ...);

   FUNCTION
    This function opens a message catalog. Catalogs contain all the
    text strings that an application uses. These strings can easily
    be replaced by strings in a different language, which causes the
    application to magically start operating in that new language.

    Catalogs originally come from disk files. This function searches for
    them in the following places:

        PROGDIR:Catalogs/languageName/name
        LOCALE:Catalogs/languageName/name

    where languageName is the name of the language associated with the
    locale parameter. So assuming an application called WizPaint:

        catalog = OpenCatalog(NULL,
                              "WizPaint.catalog",
                              OC_BuiltInLanguage,"english",
                              TAG_DONE);

    Passing NULL as first parameter to OpenCatalog() indicates you
    wish to use the system's default locale. Assuming the default locale
    specifies "deutsch" as language, OpenCatalog() tries to open the
    catalog as:

        PROGDIR:Catalogs/deutsch/WizPaint.catalog

    and if that file is not found, then OpenCatalog() tries to open it
    as:

        LOCALE:Catalogs/deutsch/WizPaint.catalog

    The OC_BuiltInLanguage tag specifies the language of the strings
    that are built into the application. If the language of the
    built-in strings matches that of the locale, then no catalog
    need be loaded from disk and the built-in strings can be used
    directly.

    locale.library caches text catalogs in order to minimize disk
    access. As such, OpenCatalog() may or may not cause disk access.
    This fact should be taken into consideration. Unused catalogs are
    automatically flushed from the system when there is not enough
    memory. When there is disk access, it is possible a DOS requester
    may be opened asking for a volume to be inserted. You can avoid this
    requester opening by setting your process' pr_WindowPtr field to -1.

   INPUTS
    locale - the locale for which the catalog should be opened, or NULL.
         When NULL, then the system's default locale is used. This
         should generally be NULL
    name - the NULL-terminated name of the catalog to open, typically
           the application name with a ".catalog" extension
    tagList - pointer to an array of tags providing optional extra
          parameters, or NULL

   TAGS
    OC_BuiltInLanguage (STRPTR) - language of built-in strings of the
                       application. That is, this tag identifies
                       the language used for the "defaultString"
                       parameter used in the GetCatalogStr()
                       function. Default is "english".

    OC_BuiltInCodeSet (ULONG) - code set of built-in strings. Default is 0.
                    THIS TAG SHOULD ALWAYS BE SET TO 0 FOR NOW.

    OC_Language (STRPTR) - language explicitly requested for the catalog.
                   A catalog of this language will be returned if
                   possible, otherwise a catalog in one of the
                   user's preferred languages. This tag should
                   normally not be provided as it overrides the
                   user's preferences.

    OC_Version (UWORD) - catalog version number required. Default is 0
                 which means to accept any version of the catalog
                 that is found. Note that if a version is
                 specified, the catalog's version much match it
                 exactly. This is different from version numbers
                 used by OpenLibrary().

   RESULTS
    catalog - a message catalog to use with GetCatalogStr() or NULL.
          A NULL result does not necessarily indicate an error.
          If OpenCatalog() determines that the built-in strings of
          the application can be used instead of an external catalog
          from disk, then NULL is returned. To determine whether
          a NULL result actually indicates an error, look at the
          return value of dos.library/IoErr(). 0 means no error.

          GetCatalogStr() interprets a NULL catalog as meaning to use
          the built-in strings.

   NOTE
    In most cases, failing to open a catalog should not be considered a
    fatal error, and the application should continue operating and
    simply use the built-in set of strings instead of the disk-based
    catalog. Note that GetCatalogStr() accepts a NULL catalog pointer for
    this very reason.

    Also note that displaying an error message when a catalog fails to
    open can be a meaningless endeavor as the message is likely in a
    language the user does not understand.

   SEE ALSO
    CloseCatalog(), GetCatalogStr()
```

```
locale.library/OpenLocale                         locale.library/OpenLocale

   NAME
    OpenLocale -- open a locale. (V38)

   SYNOPSIS
    locale = OpenLocale(name);
    D0                  A0

    struct Locale *OpenLocale(STRPTR);

   FUNCTION
    This function opens a named locale. Locales contain many parameters
    that an application needs to consider when being integrated into
    different languages, territories and customs. Using the information
    stored in a locale instead of hard-coding it into an application,
    lets the application dynamically adapt to the user's environment.

    Locales originally come from disk files. This function searches for
    them as:

        name
        SYS:Prefs/Presets/name

    Every locale specifies a language, and special language files
    must be loaded from disk depending on which language is being used.
    These files include for example:

        LOCALE:Languages/francais.language
        LOCALE:Languages/dansk.language
        LOCALE:Languages/italiano.language

   INPUTS
    name - the NULL-terminated name of the locale to open, or NULL to open
           the current default locale. This should generally be NULL.

   RESULTS
    locale - a pointer to an initialized Locale structure, or NULL if the
             locale could not be loaded. In the case of a NULL return, the
             DOS IoErr() function can be called to obtain more information
             on the failure.

             When passing a NULL name parameter to this function, you are
             guaranteed a valid return.

   SEE ALSO
    CloseLocale(), <libraries/locale.h>
```

```
locale.library/ParseDate                           locale.library/ParseDate

   NAME
    ParseDate -- interpret a string according to the date formatting
                 template and convert it into a DateStamp. (V38)

   SYNOPSIS
    state = ParseDate(locale,date,template,getCharFunc);
    D0                A0     A1   A2       A3

    BOOL ParseDate(struct Locale *,struct DateStamp *,STRPTR,struct Hook *);

   FUNCTION
    This function converts a stream of characters into an AmigaDOS
    DateStamp structure. The characters are obtained from the
    getCharFunc callback hook and the formatting template is used
    as a grammar for the parser.

   INPUTS
    locale - the locale to use for the formatting
    date - place to put the converted date, this may be NULL in which
           case this routine can be used to simply validate a date
    template - the date template describing the expected format of the
           data. See FormatDate() above for a description of date
           templates.
    getCharFunc - a callback hook invoked whenever a character is required.
              The hook should return the next character to process,
              with a NULL character to indicate the end of the string.
              The hook is called with:

                A0 - address of Hook structure
                       A1 - locale pointer
                A2 - NULL

                   The hook returns the character to process in D0. Note
              that a complete 32-bit result is expected in D0, not
              just 8 bits.

   RESULTS
    state - TRUE if the parsing went OK, or FALSE if the input did not
        match the template

   SEE ALSO
    FormatDate(), <dos/dos.h>
```

```
locale.library/StrConvert                       locale.library/StrConvert

   NAME
    StrConvert -- transform a string according to collation information.
              (V38)

   SYNOPSIS
    length = StrConvert(locale,string,buffer,bufferSize,type);
    D0                  A0     A1     A2     D0         D1

    ULONG StrConvert(struct Locale *,STRPTR,APTR,ULONG,ULONG);

   FUNCTION
    This function transforms the passed string and places the resulting
    into the supplied buffer. No more than bufferSize bytes are copied
    into the buffer.

    The transformation is such that if the C strcmp() function is applied
    to two transformed strings, it returns a value corresponding to
    the result returned by the StrnCmp() function applied to the two
    original strings.

   INPUTS
    locale - the locale to use for the transformation
    string - NULL-terminated string to transform
    buffer - buffer where to put the transformed string
    bufferSize - maximum number of bytes to deposit in the buffer
                 StrConvert() may require more storage than
             the unconverted string does
    type - describes how the transformation is to be performed. See
           the documentation on StrnCmp() for more information on the
           comparison types available

   RESULTS
    length - length of the transformed string which is the number of bytes
           deposited in the buffer minus 1 (since strings are NULL-
           terminated)

   SEE ALSO
    StrnCmp(), <libraries/locale.h>
```

```
locale.library/StrnCmp                                  locale.library/StrnCmp

   NAME
    StrnCmp -- localized string comparison. (V38)

   SYNOPSIS
    result = StrnCmp(locale,string1,string2,length,type);
    D0               A0     A1      A2      D0     D1

    LONG StrnCmp(struct Locale *,STRPTR,STRPTR,LONG,ULONG);

   FUNCTION
    Compares string1 to string2 according to the collation information
    provided by the locale and returns an integer greater than,
    equal to, or less than zero, accordingly as the string pointed to
    by string1 is greater than, equal to, or less than the string
    pointed to by string2.

    The length parameter specifies how many characters to compare, or if
    the length is specified as -1 then the strings are compared until
    a NULL is encountered.

    The type parameter dictates how the comparison is to be performed.

   INPUTS
    locale - the locale to use for this comparison
    string1 - NULL-terminated string
    string2 - NULL-terminated string
    length - the maximum number of characters to be compared, or -1 to
          compare all characters until a NULL is encountered
    type - describes how the comparison is to be performed. The following
            values can be passed:
          SC_ASCII causes an ASCII-based case-insensitive comparison
          to be performed. SC_ASCII is the fastest of the comparison
          types, but it uses ASCII ordering and considers accented
          characters different than their non-accented counterparts.

          SC_COLLATE1 causes the characters to be compared using their
          primary sorting order. This effectively produces a comparison
          that ignores letter case and diacritical marks. That is,
          letters such as "e" and "é" are treated as if they were both
          "e".

          SC_COLLATE2 causes the characters to be compared using both
          their primary and secondary sorting order. SC_COLLATE2 is
          slower than SC_COLLATE1. This is the type of comparison to
          use when sorting data to be presented to the user. It operates
          in two passes. First it performs a comparison equivalent to
          SC_COLLATE1. If both strings compare the same, then a second
          pass is made using the secondary sorting order, which gives
          finer resolution to the comparison. For example, SC_COLLATE1
          would return the following strings as identical:
             "père"  and  "pere"
          since SC_COLLATE1 ignores diacritical marks. SC_COLLATE2
          would make a second pass over the string comparing
          diacritical marks instead of actual characters.

   RESULTS
    result - relationship between string1 and string2
            <0 means string1 < string2
            =0 means string1 = string2
            >0 means string1 > string2

   SEE ALSO
    OpenLocale(), CloseLocale(), StrConvert()
```

Low-level Networking: SANA II

by Randell Jesup - Senior Software Engineer

Internet: jesup@cbmvax.cbm.commodore.com or uunet!cbmvax!jesup
BIX: rjesup

The SANA-II device specification represents a Data Link Level interface standard for Amigas. The intent is:

1) for all Amiga network hardware vendors to create a SANA-II device driver for their hardware,
2) for all protocol stack writers to access the networks via SANA-II device drivers,
3) any protocol stack to work with any SANA-II device driver.

For example, this would mean that a TCP/IP package could work with any network hardware, without modification. This should make it far easier for 3rd parties to produce networking products: hardware, protocols, and most importantly applications. An application writer would know that people could use his application without having to have a specific manufacturer's hardware board - any SANA-II device with the protocol package he used would work.

SANA-II devices are fairly normal exec devices, with an extended IORequest structure. They have a number of extended commands for handling things like network statistics, broadcast/multicast, addressing, and dealing with unexpected packets.

An integral part of SANA-II is the *netbuff.library*. It provides a method which can help avoid excessive copy of data as it is transferred from layer to layer, with ''envelopes'' of information added or removed from around it. It can be useful in layers of a protocol above the SANA-II layer as well.

Netbuff.library maintains the free pool of NetBuffSegments for use by the various network modules. It also contains several utility functions for dealing with NetBuffs as a data structure for "holding" network packet data as well as allocators and deallocators for NetBuffSegments.

The intent is for all of the network modules to use a common pool of buffers so as to reduce the memory requirements of the network software, and for the network modules to be able to pass network data (packets) with minimal copying. There may be some changes to how netbuffs are implemented by the time you read this, contact CATS or the SANA-II networking group for more information.

The SANA-II specification is an evolving document, though it's fairly well defined at this point. There are a few changes that may occur in the near future regarding protocol/network/interface dependent portions of the spec, and the station aliases. Other small changes may occur as implementations by various people progress. Radical changes are unlikely at this point.

Goals

Simplicity: A SANA-II device driver should be as simple to write as any other Amiga device.

Low cost: A functional SANA-II device driver should take about one man month or less for an experienced Amiga programmer with Amiga device writing experience to write. A tuned and polished SANA-II device should require only one additional man month.

Compatibility: All SANA-II device drivers behave largely the same.

Low resource usage: A SANA-II device driver should not require large amounts of memory, cpu cycles or any other system resource to operate efficiently.

Ease of use: Using SANA-II devices should be the easiest way for protocol stack writers to access the networks.

Non-Goals

Lowest common denominator: Not all network interfaces can support all the features available in this specification or other network types.

Feature-itis: Not every feature of every type of network is represented.

Network management tools: Analysis and debugging of a network requires specialized software for that purpose.

Network/Interface Dependant Items:

A few parts of SANA-II are dependent on the type of network/interface being used. These include some statistics, Broadcast and Multicast packets, and the method for determining packet type. Packet type is currently handled by the Magic field. This may be changed by the time of the Denver DevCon talk (or soon thereafter) to use of a callback to the protocol stack, so devices won't have to have any knowledge of how packet type is encoded. Something similar may be done for Broadcast and Multicast.

It is also possible to access the device in a "raw" fashion for protocols or other modules, such as network management, etc, which need access to the raw packet data.

Structure Of A Sana-II Device Driver:

SANA-II drivers are very similar to other Amiga device drivers. The general concept involves posting reads for packets you're interested in, and writing packets as needed. The SANA2CMD_DEVICEQUERY command returns device data used by a protocol, such as the maximum packet data size and what type of physical network is being used (in case the protocol has some network-specific code for, as an example, multicasts. SANA2CMD_ONEVENT allows the protocol to wait for a specific condition to occur, and get notification of that condition. The protocol may wish to keep an SANA2CMD_READORPHAN active if it wishes to know of packets which are of types not being waited on. This might include RAW packets for network- specific portions of a protocol. Lastly, there are many types of statistics about packets received and sent that can be gathered.

In terms of handling requests, it's a normal Amiga device, and all the normal flags such as IOF_QUICK apply. The iorequest used is (in a similar manner to the *serial.device*) extended with a number of additional fields.

For information on how to construct an Exec device, consult the *ROM Kernel Manual* example of a device. Hopefully we should have some example code for writing SANA-II devices and using SANA-II from a protocol available in the near future.

```
sana2.device/AbortIO()

    NAME
        AbortIO -- Remove an existing device request.

    SYNOPSIS
        error = AbortIO( Sana2Req )
        D0               A1

        LONG AbortIO( struct IOSana2Req * );

    FUNCTION
        This is an exec.library call.

        This function aborts an ioRequest. If the request is active,
        it may or may not be aborted. If the request is queued it is
        removed. The request will be returned in the same way as if
        it had normally completed.

    INPUTS
        Sana2Req        Sana2Req to be aborted.

    RESULTS
        error           Zero if the request was aborted, non-zero
                            otherwise. S2io_Error in Sana2Req will be
                            set to IOERR_ABORTED (-2) if it was
                            aborted.

    NOTES

    SEE ALSO
        exec.library/AbortIO()

    BUGS
```

```
sana2.device/CMD_READ

    NAME
        Read -- Get a packet from the network.

    FUNCTION
        Get the next packet available of the requested packet type.
        The data returned in the S2io_Body NetBuff is normally the
        Data Link Layer packet data only. If bit SANA2IOB_RAW is set
        in S2io_Flags, then the entire Data Link Layer packet,
        including both header and trailer information, will be
        returned.

    IO REQUEST
        S2io_Command    CMD_READ
        S2io_Flags      Supported flags are:
                            SANA2IOB_RAW
                            SANA2IOB_QUICK
        S2io_PacketType Pointer to packet type desired.
        S2io_Body       NetBuff to hold packet data.

    RESULTS
        S2io_Error      Zero if successful; non-zero otherwise.
        S2io_WireError  More specific error number.
        S2io_Flags      The following flags may be returned:
                            SANA2IOB_RAW
                            SANA2IOB_BCAST
                            SANA2IOB_MCAST
        S2io_SrcAddr    Source interface address of packet.
        S2io_DstAddr    Destination interface address of packet.
        S2io_DTLLength  Length of packet data as given by the Data
                            Link Layer protocol.
        S2io_Body       NetBuff with packet data.

    NOTES
        Only the first NetBuffSegment in the NetBuff passed is
        guaranteed to be returned.

    SEE ALSO
        sana2.device/SANA2CMD_READORPHAN,
        sana2.device/CMD_WRITE

    BUGS
```

```
sana2.device/CMD_WRITE

   NAME
        Write -- Send packet to the network.

   FUNCTION
        This command causes the packet to be sent to the specified
        network interface. Normally, appropriate packet header and
        trailer information will be added to the packet data when it
        is sent. If bit SANA2IOB_RAW is set in io-Flags, then the
        S2io_Body NetBuff is assumed to contain an entire Data Link
        Layer packet and will be sent unmodified.

        The data in the S2io_Body NetBuff is returned unmodified; but,
        the distribution of the data in, and the number of
        NetBuffSegments, the S2io_Body NetBuff may be modified.

   IO REQUEST
        S2io_Command     CMD_WRITE
        S2io_Flags       Supported flags are:
                             SANA2IOB_RAW
                             SANA2IOB_QUICK
        S2io_PacketType Pointer to type of packet to send.
        S2io_DstAddr     Destination interface address for this packet.
        S2io_Body        NetBuff with packet data.

   RESULTS
        S2io_Error       Zero if successful; non-zero otherwise.
        S2io_WireError   More specific error number.

   NOTES

   SEE ALSO
        sana2.device/CMD_READ,
        sana2.device/SANA2CMD_BROADCAST,
        sana2.device/SANA2CMD_MULTICAST

   BUGS
```

```
sana2.device/SANA2CMD_ADDMULTICASTADDRESS

   NAME
        AddMulticastAddress -- Enable an interface multicast address.

   FUNCTION
        This command causes the device driver to enable multicast
        packet reception for the requested address.

   IO REQUEST
        S2io_Command     SANA2CMD_ADDMULTICASTADDRESS
        S2io_SrcAddr     Multicast address to enable.

   RESULTS
        S2io_Error       Zero if successful; non-zero otherwise.
        S2io_WireError   More specific error number.

   NOTES
        Since multicast addresses are not "bound" to a particular
        packet type, each enabled multicast address has an "enabled"
        count associated with it.

   SEE ALSO
        sana2.device/SANA2CMD_DELMULTICASTADDRESS

   BUGS
```

```
sana2.device/SANA2CMD_ADDSTATIONALIAS

   NAME
        AddStationAlias -- Add interface address alias.

   FUNCTION
        This function adds another network interface address to which
        the network interface should respond.

   IO REQUEST
        S2io_Command    SANA2CMD_ADDSTATIONALIAS
        S2io_SrcAddr    Alias to add.

   RESULTS
        S2io_Error      Zero if successful; non-zero otherwise.
        S2io_WireError  More specific error number.

   NOTES
        Since alias interface addresses are not bound to a particular
        packet type or read request, each added interface alias
        address has an "added" count associated with it.

   SEE ALSO
        sana2.device/SANA2CMD_DELSTATIONALIAS

   BUGS
```

```
sana2.device/SANA2CMD_BROADCAST

   NAME
        Broadcast -- Broadcast a packet on network.

   FUNCTION
        This command works the same as CMD_WRITE except that it also
        performs whatever special processing of the packet is required
        to do a broadcast send. The actual broadcast mechanism is
        neccessarily network/interface/device specific.

   IO REQUEST
        S2io_Command    SANA2CMD_BROADCAST
        S2io_Flags      Supported flags are:
                            SANA2IOB_RAW
                            SANA2IOB_QUICK
        S2io_PacketType Pointer to type of packet to send.
        S2io_Body       NetBuff with packet data.

   RESULTS
        S2io_DstAddr    The contents of this field are to be
                            considered trash upon return of the IOReq.
        S2io_Error      Zero if successful; non-zero otherwise.
                            This command can fail for many reasons and
                            is not supported by all networks and/or
                            network interfaces.
        S2io_WireError  More specific error number.

   NOTES

   SEE ALSO
        sana2.device/CMD_WRITE,
        sana2.device/SANA2CMD_MULTICAST

   BUGS
```

```
sana2.device/SANA2CMD_CONFIGINTERFACE

   NAME
        ConfigInterface -- Configure the network interface.

   FUNCTION
        This command causes the device driver to initialize the
        interface hardware and to set the network interface address to
        the address in S2io_SrcAddr. This command can only be executed
        once and, if successful, will leave the driver and network
        interface fully operational and the network interface in
        S2io_SrcAddr.

        To set the interface address to the factory address, the
        network management software must use GetStationAddress first
        and then call ConfigInterface with the result. If there is no
        factory address then the network software must pick an address
        to use.

        Until this command is executed the device will not listen for
        any packets on the hardware.

   IO REQUEST
        S2io_Command    SANA2CMD_CONFIGINTERFACE
        S2io_Flags      Supported flags are:
                            SANA2IOB_QUICK
        S2io_SrcAddr    Address for this interface.

   RESULTS
        S2io_Error      Zero if successful; non-zero otherwise.
        S2io_WireError  More specific error number.
        S2io_SrcAddr    Address of this interface as configured.

   NOTES
        Some networks have the interfaces choose a currently unused
        interface address each time the interface is initialized. The
        caller must check S2io_SrcAddr for the actual interface
        address after configuring the interface.

   SEE ALSO
```

```
        sana2.device/SANA2CMD_GETSTATIONADDRESS

   BUGS
        What to do when/if the network interface address changes due
        to a network reconfiguration.
```

```
sana2.device/SANA2CMD_DELMULTICASTADDRESS

   NAME
        DelMultiCastAddress -- Disable an interface multicast address.

   FUNCTION
        This command causes device driver to disable multicast packet
        reception for the requested address.

        It is an error to disable a multicast address that is not
        enabled.

   IO REQUEST
        S2io_Command    SANA2CMD_DELMULTICASTADDRESS
        S2io_SrcAddr    Multicast address to disable.

   RESULTS
        S2io_Error      Zero if successful; non-zero otherwise.
        S2io_WireError  More specific error number.

   NOTES
        Since multicast addresses are not "bound" to a particular
        packet type, each enabled multicast address has an "enabled"
        count associated with it.

   SEE ALSO
        sana2.device/SANA2CMD_ADDMULTICASTADDRESS

   BUGS
```

```
sana2.device/SANA2CMD_DELSTATIONALIAS

   NAME
        DelStationAlias -- Delete a interface address alias.

   FUNCTION
        This function deletes a network interface address from the
        list of addresses to which this interface should respond.

   IO REQUEST
        S2io_Command     SANA2CMD_DELSTATIONALIAS
        S2io_SrcAddr     Alias to delete.

   RESULTS
        S2io_Error       Zero if successful; non-zero otherwise.
        S2io_WireError   More specific error number.

   NOTES
        Since alias interface addresses are not bound to a particular
        packet type or read request, each added interface alias
        address has an "added" count associated with it.

   SEE ALSO
        sana2.device/SANA2CMD_ADDSTATIONALIAS

   BUGS
```

```
sana2.device/SANA2CMD_DEVICEQUERY

   NAME
        DeviceQuery -- Return parameters for this network interface.

   FUNCTION
        This command causes the device driver to report information
        about the device. Up to DevQuerySizeAvailable bytes of the
        information is copied into a buffer pointed to by
        S2io_StatData. The format of the data is as follows:

             struct Sana2DeviceQuery
                 {
                 /*
                  * Standard information
                  */
                 ULONG DevQuerySizeAvailble; /* bytes available */
                 ULONG DevQuerySizeSupplied; /* bytes supplied */
                 LONG  DevQueryFormat;       /* this is type 0 */
                 LONG  DeviceLevel;          /* this document is level 0 */

                 /*
                  * Common information
                  */
                 UWORD AddrFieldSize;        /* address size in bits */
                 ULONG MTU;                  /* maximum packet data size */
                 LONG  bps;                  /* line rate (bits/sec) */
                 LONG  HardwareType;         /* what the wire is */

                 /*
                  * Format specific information
                  */
                 };

        The DeviceQueryAvailable specifies the number of bytes that
        the caller is prepared to accomodate, including the standard
        information fields.

        DevQuerySizeSupplied is the number of bytes actually supplied,
        including the standard information fields, which will not
        exceed DevQuerySizeAvailable.

        The values used to identify the type of physical hardware are
        the same values used in the hardware type field (hdr) of
        Address Resolution Protocol (ARP) packets as defined by
        RFC-826 (ARP) and RFC-1060 (Assigned Numbers, March 1990).

        The following are defined in RFC-1060.

            Hardware Type (hrd)

                Type   Description
                ----   -----------
                  1    Ethernet (10Mb)
                  2    Experimental Ethernet (3Mb)
                  3    Amateur Radio AX.25
                  4    Proteon ProNET Token Ring
                  5    Chaos
                  6    IEEE 802 Networks
                  7    ARCNET
                  8    Hyperchannel
                  9    Lanstar
                 10    Autonet Short Address
                 11    LocalTalk
                 12    LocalNet (IBM PCNet or SYTEK LocalNET)

   IO REQUEST
        S2io_Command     SANA2CMD_DEVICEQUERY
        S2io_StatData    Pointer to Sana2DeviceQuery structure to fill
                            in.

   RESULTS
        S2io_Error       Zero if successful; non-zero otherwise.
        S2io_WireError   More specific error number.

   NOTES
   SEE ALSO

   BUGS


sana2.device/SANA2CMD_GETGLOBALSTATS

   NAME
        GetGlobalStats -- Get interface accumulated statistics.

   FUNCTION
        This command causes the device driver to retrieve various
        global runtime statistics for this network interface. The
        format of the data returned is as follows:

            struct Sana2DeviceStats
                {
                LONG packets_received;
                LONG packets_sent;
                LONG framming_errors;
                LONG bad_data;
                LONG bytes_received;
                LONG bytes_sent;
                LONG hard_misses;
                LONG soft_misses;
                LONG unknown_types_received;
                LONG fifo_overruns;
                LONG fifo_underruns;
                LONG reconfigurations;
                timeval last_start;
                };

   IO REQUEST
        S2io_Command    SANA2CMD_GETGLOBALSTATS
        S2io_StatData   Pointer to Sana2DeviceStats structure to fill.

   RESULTS
        S2io_Error      Zero if successful; non-zero otherwise.
        S2io_WireError  More specific error number.

   NOTES

   SEE ALSO
        sana2.device/SANA2CMD_GETSPECIALSTATS

   BUGS
```

```
sana2.device/SANA2CMD_GETSPECIALSTATS

   NAME
        GetSpecialStats -- Get network type specific statistics.

   FUNCTION
        This function returns statistics which are specific to the
        type of network medium this driver controls. For example, this
        command could return statistics common to all Ethernets which
        are not common to all network mediums in general.

        The supplied Sana2SpecialStatData structure is given below:

            struct Sana2SpecialStatData
                {
                ULONG RecordCountMax;
                ULONG RecordCountSupplied;
                struct Sana2StatRecord[RecordCountMax];
                };

        The format of the data returned is:

            struct Sana2StatRecord
                {
                ULONG Type;     /* commodore registered */
                LONG Count;     /* the stat itself */
                char *String;   /* null terminated */
                };

        The RecordCountMax field specifies the number of records that
        the caller is prepared to accomodate.

        RecordCountSupplied is the number of record actually supplied
        which will not exceed RecordCountMax.

   IO REQUEST
        S2io_Command    SANA2CMD_GETSPECIALSTATS
        S2io_StatData   Pointer to a Sana2SpecialStatData structure to
                            fill. RecordCountMax must be initialized.

   RESULTS
        S2io_Error      Zero if successful; non-zero otherwise.
        S2io_WireError  More specific error number.

   NOTES
        Commodore shall maintain registered statistic Types.

   SEE ALSO
        sana2.device/SANA2CMD_GETGLOBALSTATS

   BUGS
```

```
sana2.device/SANA2CMD_GETSTATIONADDRESS

   NAME
        GetStationAddress -- Get default and interface address.

   FUNCTION
        This command causes the device driver to copy the current
        interface address into S2io_SrcAddr, and to copy the factory
        default station address into S2io_DstAddr.

   IO REQUEST
        S2io_Command     SANA2CMD_GETSTATIONADDRESS

   RESULTS
        S2io_Error       Zero if successful; non-zero otherwise.
        S2io_WireError   More specific error number.
        S2io_SrcAddr     Default interface address.
        S2io_DstAddr     Current interface address.

   NOTES

   SEE ALSO
        sana2.device/SANA2CMD_CONFIGINTERFACE

   BUGS
```

```
sana2.device/SANA2CMD_GETTYPESTATS

   NAME
        GetTypeStats -- Get accumulated type specific statistics.

   FUNCTION
        This command causes the device driver to retrieve various
        packet type specific runtime statistics for this network
        interface. The format of the data returned is as follows:

            struct Sana2TypeStatData
                {
                LONG PacketsSent;
                LONG PacketsReceived;
                LONG BytesSent;
                LONG BytesReceived;
                LONG PacketsDropped;
                };

   IO REQUEST
        S2io_Command     SANA2CMD_GETTYPESTATS
        S2io_PacketType  Pointer to the packet type of interest.
        S2io_StatData    Pointer to TypeStatData structure to fill in.

   RESULTS
        S2io_Error       Zero if successful; non-zero otherwise.
        S2io_WireError   More specific error number.

   NOTES
        Statistics for a particular packet type are only available
        while that packet type is being ''tracked''.

   SEE ALSO
        sana2.device/SANA2CMD_TRACKTYPE,
        sana2.device/SANA2CMD_UNTRACKTYPE

   BUGS
```

```
sana2.device/SANA2CMD_MULTICAST

   NAME
        Multicast -- Multicast a packet on network.

   FUNCTION
        This command works the same as CMD_WRITE except that it also
        performs whatever special processing of the packet is required
        to do a multicast send. The actual multicast mechanism is
        neccessarily network/interface/device specific.

   IO REQUEST
        S2io_Command     SANA2CMD_MULTICAST
        S2io_Flags       Supported flags are:
                             SANA2IOB_RAW
                             SANA2IOB_QUICK
        S2io_PacketType  Pointer to type of packet to send.
        S2io_DstAddr     Destination interface address for this packet.
        S2io_Body        NetBuff with packet data.

   RESULTS
        S2io_Error       Zero if successful; non-zero otherwise.
                             This command can fail for many reasons and
                             is not supported by all networks and/or
                             network interfaces.
        S2io_WireError   More specific error number.

   NOTES
        The address supplied in S2io_DstAddr will be sanity checked
        (if possible) by the driver. If the supplied address fails
        this sanity check, the multicast request will fail
        immediately.

   SEE ALSO
        sana2.device/CMD_WRITE,
        sana2.device/SANA2CMD_BROADCAST

   BUGS
```

```
sana2.device/SANA2CMD_OFFLINE

   NAME
        Offline -- Remove interface from service.

   FUNCTION
        This command removes a network interface from service.

   IO REQUEST
        S2io_Command    SANA2CMD_OFFLINE

   RESULTS
        S2io_Error      Zero if successful; non-zero otherwise.
        S2io_WireError  More specific error number.

   NOTES
        Aborts all pending reads and writes with S2io_Error set to
        SANA2ERROR_OUTOFSERVICE.

        While the interface is offline, all read, writes and any other
        command that touches interface hardware will be rejected with
        S2io_Error set to SANA2ERROR_OUTOFSERVICE.

        This command is intended to permit a network interface to be
        tested on an otherwise live system.

   SEE ALSO
        sana2.device/SANA2CMD_ONLINE

   BUGS
```

```
sana2.device/SANA2CMD_ONEVENT

   NAME
        OnEvent -- Return when specified event occures.

   FUNCTION
        This command returns when a particular event condition has
        occured on the network or this network interface.

   IO REQUEST
        S2io_Command    SANA2CMD_ONEVENT
        S2io_Flags      Supported flags are:
                            SANA2IOB_QUICK
        S2io_WireError  Event number to wait for.

   RESULTS
        S2io_Error      Zero if successful; non-zero otherwise.
        S2io_WireError  Event number that occured.

   NOTES
        If this device driver does not understand the specified event
        condition then the command returns immediately with S2io_Error
        non-zero. A successful return will have S2io_Error set to zero
        S2io_WireError set to the event number.

        All pending requests for a particular event will be returned
        when that event occurs.

        All event types that cover a particular condition are returned
        when that condition occures. For instance, if an error is
        returned by NetBuff.library during receive processing, events
        of types S2EVENT_ERROR, S2EVENT_RX and S2EVENT_NETBUFF would
        be returned if pending.

   SEE ALSO

   BUGS
```

```
sana2.device/SANA2CMD_ONLINE

   NAME
        Online -- Put a network interface back in service.

   FUNCTION
        This command places an offline network interface back into
        service.

   IO REQUEST
        S2io_Command     SANA2CMD_ONLINE

   RESULTS
        S2io_Error       Zero if successful; non-zero otherwise.
        S2io_WireError   More specific error number.

   NOTES
        This command is responsible for putting the network interface
        hardware back into a known state and resets the unit global
        and special statistics.

   SEE ALSO
        sana2.device/SANA2CMD_OFFLINE

   BUGS
```

```
sana2.device/SANA2CMD_READORPHAN

   NAME
        ReadOrphan -- Get a packet for which there is no reader.

   FUNCTION
        Get the next packet available that does not satisfy any
        then-pending CMD_READ requests. The data returned in the
        S2io_Body NetBuff is normally the Data Link Layer packet type
        field and the packet data. If bit SANA2IOB_RAW is set in
        S2io_Flags, then the entire Data Link Layer packet, including
        both header and trailer information, will be returned.

   IO REQUEST
        S2io_Command     CMD_READORPHAN
        S2io_Flags       Supported flags are:
                             SANA2IOB_RAW
                             SANA2IOB_QUICK
        S2io_Body        NetBuff to hold packet data.

   RESULTS
        S2io_Error       Zero if successful; non-zero otherwise.
        S2io_WireError   More specific error number.
        S2io_Flags       The following flags may be returned:
                             SANA2IOB_RAW
                             SANA2IOB_BCAST
                             SANA2IOB_MCAST
        S2io_SrcAddr     Source interface address of packet.
        S2io_DstAddr     Destination interface address of packet.
        S2io_DTLLength   Lenght of packet data as given by the Data
                             Link Layer protocol.
        S2io_Body        NetBuff with packet data.

   NOTES
        Only the first NetBuffSegment in the NetBuff passed in is
        guarenteed to be returned.

        To determine which protocol family the returned packet belongs
        to you may have to specify SANA2IOB_RAW to get the entire data
        link layer wrapper (which is where the protocol type may be
        kept). Notice this necessarily means that this cannot be done
        in a network interface independent fashion.

   SEE ALSO
        sana2.device/CMD_READ,
        sana2.device/CMD_WRITE

   BUGS
```

```
sana2.device/SANA2CMD_TRACKTYPE

   NAME
        TrackType -- Accumulate statistics about a packet type.

   FUNCTION
        This command causes the device driver to accumulate statistics
        about a particular packet type. Packet type statistics, for
        the particular packet type, are zeroed by this command.

   IO REQUEST
        S2io_Command    SANA2CMD_TRACKTYPE
        S2io_PacketType Pointer to the packet type of interest.

   RESULTS
        S2io_Error      Zero if successful; non-zero otherwise.
        S2io_WireError  More specific error number.

   NOTES

   SEE ALSO
        sana2.device/SANA2CMD_UNTRACKTYPE,
        sana2.device/SANA2CMD_GETTYPESTATS

   BUGS
```

```
sana2.device/SANA2CMD_UNTRACKTYPE

   NAME
        UntrackType -- End statistics about a packet type.

   FUNCTION
        This command causes the device driver to stop accumulating
        statistics about a particular packet type.

   IO REQUEST
        S2io_Command    SANA2CMD_UNTRACKTYPE
        S2io_PacketType Pointer to the packet type of interest.

   RESULTS
        S2io_Error      Zero if successful; non-zero otherwise.
        S2io_WireError  More specific error number.

   NOTES

   SEE ALSO
        sana2.device/SANA2CMD_TRACKTYPE,
        sana2.device/SANA2CMD_GETTYPESTATS

   BUGS


   STRUCTURES
            struct IOSana2Req
                {
                struct Message S2io_Message;
                struct Device *S2io_Device;
                struct Unit   *S2io_Unit;

                UWORD S2io_Command;
                UBYTE S2io_Flags;
                BYTE  S2io_Error;
                ULONG S2io_WireError;

                struct Sana2PacketType *S2io_PacketType;
                UBYTE S2io_SrcAddr[SANA2_MAX_ADDR_BYTES];
                UBYTE S2io_DstAddr[SANA2_MAX_ADDR_BYTES];
                ULONG S2io_DataLength;
                struct NetBuff S2io_Body;
                void *S2io_StatData;
                };

            Message
                Initialized and used the same as all device IORequest
                Message structures.

            Device
                Device private.

            Unit
                Device private.

            Command
                Command requested.

            Flags
                Used to request special options and return status
                information.

            Error
                Indicates completion status of a command in general
                terms.

            WireError
                More specific error number. Only valid when S2io_Error
                is non-zero.

            PacketType
                Pointer to a Sana2PacketType structure. The device
                uses the contents of the structure to determine if a
                packet from the network will satisfy a particular read
                command, or to build the appropriate packet header and
```

trailer for a packet send. The device will not modify either the pointer or the structure.

SrcAddr
The device fills in this field with the interface address of the source of the packet that satisfied a read command. The bytes used to hold the address will be left justified but the bit layout is dependent on the particular type of network.

DstAddr
This field is filled in with the interface destination address of the packet for a send command. The bytes used to hold the address will be left justified but the bit layout is dependent on the particular type of network.

DataLength
The device fills in this field with the size of the packet data as it was sent on the wire. This does not include the header and trailer information. Depending on the network type and protocol type, the driver may have to calculate this value.

Body
A NetBuff structure with the packet data.

StatData
Pointer to a data area in memory to place a a snapshot of device statics. The data area must be long word aligned.

```
struct Sana2PacketType
    {
    ULONG CanonicalType;
    ULONG Magic;
    ULONG Length;
    UBYTE *Match;
    UBYTE *Mask;
    };
```

CanonicalType
Not used by the device.

Magic
Determines how the fields in this structure are to be interpretted.

Length
Number of bytes pointed to be Match and Mask.

Match
Pointer to packet type match data. The data must be long word (32 bit) aligned.

Mask
Pointer to packet type mask data. The data must be long word (32 bit) aligned.

```
struct Sana2DeviceQuery
    {
    /*
     * Standard information
     */
    ULONG SizeAvailable;
    ULONG SizeSupplied;
    LONG DevQueryFormat;
    LONG DeviceLevel;
    /*
     * Common information
     */
    UWORD AddrFieldSize;
    ULONG MTU;
    LONG bps;
    LONG HardwareType;
    /*
     * Format specific information
     */
    };
```

SizeAvailable
Size, in bytes, of the space available in which to place device information. This includes both size fields.

SizeSupplied
Size, in bytes, of the data supplied.

DevQueryFormat
The format defined here is format 0.

DeviceLevel
This spec defines level 0.

AddrFieldSize
The number of bits in an interface address.

MTU
Maximum Transmission Unit, the size, in bytes, of the maximum packet size, not including header and trailer information.

bps
Best guess at the raw line rate for this network in bits per second.

HardwareType
Specifies the type of network wire the driver controls.

```
struct Sana2PacketTypeStats
    {
    LONG PacketsSent;
    LONG PacketsReceived;
    LONG BytesSent;
    LONG BytesReceived;
    LONG PacketsDropped;
    };
```

PacketsSent
Number of packets of a particular type sent.

PacketsReceived
Number of packets of a particular type that satisfied a read command.

BytesSent
Number of bytes of data sent in packets of a particular type.

BytesReceived
Number of bytes of data of a particular packet type that satisfied a read command.

PacketsDropped
Number of packets of a particular type that were received while there were no pending reads of that packet type.

```
struct Sana2SpecialStatRecord
    {
    ULONG Type;
    LONG Count;
    char *String;
    };
```

Type
Statistic identifier.

Count

```
        Statistic itself.

    String
        Identifying string for the statistic. Should be plain
        ASCII with no formatting characters.


    struct Sana2SpecialStatHeader
        {
        ULONG RecordCountMax;
        ULONG RecordCountSupplied;
        struct Sana2SpecialStatRecord[RecordCountMax];
        };

    RecordCountMax
        Space for this many number of records is available to
        place statistics information in.

    RecordCountSupplied
        Number of statistic records supplied.


    struct Sana2DeviceStats
        {
        LONG packets_received;
        LONG packets_sent;
        LONG framing_errors;
        LONG bad_data;
        LONG hard_misses;
        LONG soft_misses;
        LONG unknown_types_received;
        LONG fifo_overruns;
        LONG fifo_underruns;
        LONG reconfigurations;
        struct timeval last_start;
        };

    packets_received
        Number of packets that this unit has received.

    packets_sent
        Number of packets that this unit has sent.

    framing_errors
        Number of framing errors detected.

    bad_data
        Number of bad packets received.

    hard_misses
        Number of packets lost due to insufficient resources
        available in the network interface.

    soft_misses
        Number of packets lost due to insifficient resources
        available on the host.

    unknown_type_received
        Number of packets received that had no pending read
        command with the appropriate packet type.

    fifo_overruns
        Number of overruns encountered when attempting to
        store received packets.

    fifo_underruns
        Number of underruns encountered when attempting to
        send packets.

    reconfigurations
        Number of network reconfigurations since this unit was
        last configured.

    last_start
        The time when this unit last went online.
```

```
CONSTANTS
    OpenDevice flags argument (SANA2OPB_xxx):
        MINE
            Exclusive access to the unit requested.

        PROM
            Promiscuous mode requested.


    IOSana2Req S2io_Flags field (SANA2IOB_xxx):
        RAW
            Raw packet read/write requested.

        BCAST
            Broadcast packet (received).

        MCAST
            Multicast packet (received).

        QUICK
            Quick IO requested.


    IOSana2Req S2io_Error field (S2ERR_xxx):
        NO_RESOURCES
            Insufficient resources available.

        UNKNOWN_ENTITY
            Specified entity unknown or not found.

        BAD_ARGUMENT
            Noticably bad argument.

        BAD_STATE
            Command inappropriate for current state.

        BAD_ADDRESS
            Noticably bad address.

        MTU_EXCEEDED
            Write data too large.

        BAD_PROTOCOL
            Noticably bad packet type.

        NOT_SUPPORTED
            Command is not supported by this driver. This is
            similar to IOERR_NOCMD as defined in exec/errors.h but
            S2ERR_NOT_SUPPORTED indicates that the requested
            command is a valid SANA-II command and that the driver
            does not support it, e.g. SANA2CMD_MULTICAST.

        SOFTWARE
            Software error of some kind.


    IOSana2Req S2io_WireError field (S2WERR_xxx):
        NOT_CONFIGURED
            Command requires unit to be configured.

        UNIT_ONLINE
            Command requires that the unit be offline.

        UNIT_OFFLINE
            Command requires that the unit be online.

        ALREADY_TRACKED
            Protocol is already being tracked.

        NOT_TRACKED
            Protocol is not being tracked.

        NETBUFF_ERROR
            NetBuff.library error return caused error.

        SRC_ADDRESS
            Problem with the source address field.
```

DST_ADDRESS
Problem with destination address field.

BAD_BROADCAST
Problem with an attempt to broadcast.

BAD_MULTICAST
Problem with an attempt to multicast.

ALIAS_LIST_FULL
Station alias list full.

BAD_ALIAS
Bad or unknown station alias.

MULTICAST_FULL
Multicast address list full.

BAD_EVENT
Event specified is unknown.

BAD_STATDATA
The S2IO_StatData pointer or the data it points to failed a sanity check.

PROTOCOL_UNKNOWN
The protocol specified is unknown or the type of magic used by the request protocol is unknown.

IS_CONFIGURED
Attempt to reconfigure the unit.

NULL_POINTER
A NULL pointer was detected in one of the arguments.

Event types (S2EVENT_xxx):

ERROR
Return when any error occures.

TX
Return on any transmit error.

RX
Return on any receive error.

ONLINE
Return when unit goes online.

OFFLINE
Return when unit goes offline.

NETBUFF
Return on any NetBuff.library error.

HARDWARE
Return when any hardware error occures.

SOFTWARE
Return when any software error occures.

NETBUFF.LIBRARY STRUCTURES

A NetBuff is a data structure representing a logical array of of bytes.

```
struct NetBuff
    {
    struct MinList List;
    ULONG Count;
    };
```

List
List of NetBuffSegments.

Count
The number of bytes of data the NetBuff is said to contain.

A NetBuffSegment is a data structure used to store and keep track of all or part of the data in a NetBuff.

```
struct NetBuffSegment
    {
    struct MinNode Node;
    ULONG PhysicalSize;
    ULONG DataOffset;
    ULONG DataCount;
    UBYTE *Buffer;
    };
```

Node
Node structure linking the NetBuffSegments together.

PhysicalSize
The size of the data area that Buffer points to. If this field is zero, the actual size of the data area unknown and it is managed (allocated and freed) by some other entity. Only 'Count' bytes starting at '(Buffer+Offset)' are known to be valid.

DataOffset
Offset into the Buffer where the data starts.

DataCount
Number of data bytes that this NetBuffSegment contains.

Buffer
Pointer to the start of the data area for this NetBuffSegment.

PHYSICALSIZE ZERO SEGMENTS

Any software making use of NetBuffs must correctly handle NetBuffs with PhysicalSize zero segments. The general rules to follow are as follows:

+ "Send" NetBuffs, those NetBuffs where the data is created and passed to a lower network layer, are returned to the creator with the data intact. This class of NetBuffs may have segments of PhysicalSize zero.

+ "Receive" NetBuffs, those NetBuffs where the data is created and passed to a higher network layer, are not returned to the creator. This class of NetBuffs may not have segments of PhysicalSize zero.

+ Lower network layers will eventually return the actual NetBuff structure (pointer) to the layer that created the data.

+ NetBuffs returned to higher layers from lower layers may have the "physical" layout of the data changed. The layout of the "logical" data will not have changed.

+ Track NetBuff structures to know when it is safe to reuse/deallocate storage for PhysicalSize zero segments.

In to above, higher and lower network layers refers to the usual diagram of the OSI network layering model; where the application layer is at the top of the diagram and the physical layer is at the bottom.

This model for handling PhysicalSize zero segments also has obvious advantages in situations where time-outs and retransmittions occure.

```
netbuff.library/AllocSegments()

   NAME
        AllocSegments -- Get a list of NetBuffSegments.

   SYNOPSIS
        AllocSegments( Count, Segment_List )
                       D0     A0

        void AllocSegments( ULONG, struct List * );

   FUNCTION
        This function returns a list of NetBuffSegments sufficient to
        hold Count bytes.

   INPUTS
        Count           Numbers of bytes for which space is needed.
        Segment_List    Pointer to an empty (initialized) list to hold
                            the allocated NetBuffSegments.

   RESULTS

   NOTES
        The function cannot be called from interrupts.

   SEE ALSO
        netbuff.library/IntAllocSegments(),
        netbuff.library/FreeSegments(),
        netbuff.library/ReadyNetBuff()

   BUGS
```

```
netbuff.library/CompactNetBuff()

   NAME
        CompactNetBuff -- Optimize a NetBuff.

   SYNOPSIS
        error = CompactNetBuff( Netbuff )
        D0                      A0

        LONG CompactNetBuff( struct NetBuff * );

   FUNCTION
        This function optimizes a NetBuff by moving the data to fill
        each needed NetBuffSegment as much as possible and return
        unused NetBuffSegments to the free pool. This function will
        not modify NetBuffSegments of physical size zero.

   INPUTS
        Netbuff         Pointer to NetBuff structure to optimize.

   RESULTS
        error           Zero if successful; non-zero otherwise.

   NOTES
        This function consumes time.

   SEE ALSO

   BUGS
```

```
netbuff.library/CopyFromNetBuff()

    NAME
        CopyFromNetBuff -- Copy data from a NetBuff to memory.

    SYNOPSIS
        error = CopyFromNetBuff( Netbuff, Offset, Data, Count )
        D0                       A0       D0      A1    D1

        LONG CopyFromNetBuff( struct NetBuff *, LONG, UBYTE *, ULONG );

    FUNCTION
        This function copies 'Count' bytes from a NetBuff to memory
        pointed to by Data. If Offset is non-negative, then the start
        of the data to copy is 'Offset' bytes from the start of the
        NetBuff. If Offset is negative, then the start of the data to
        copy is 'abs(Offset)' bytes from the end of the NetBuff.

    INPUTS
        NetBuff         Pointer to NetBuff structure with source data.
        Offset          Offset into Netbuff where data starts.
        Data            Pointer to area to store data in.
        Count           Number of bytes to copy.

    RESULTS
        error           Zero if successful; non-zero otherwise.

    NOTES
        This function may be called from interrupts.

        This function might be used by a network device driver to
        extract bytes from a NetBuff to fill hardware.

        This function might be used within a protocol stack to extract
        data structures from a NetBuff into local memory.

    SEE ALSO
        netbuff.library/CopyToNetBuff()

    BUGS
```

```
netbuff.library/CopyNetBuff()

    NAME
        CopyNetBuff -- Make a copy of a NetBuff.

    SYNOPSIS
        error = CopyNetBuff( Netbuff0, Netbuff1 )
        D0                   A0        A1

        LONG CopyNetBuff( struct NetBuff *, struct NetBuff * );

    FUNCTION
        This function makes NetBuff1 a logical clone of Netbuff0.
        Netbuff0 will be unmodified and Netbuff1 will be optimal.

        All data in Netbuff1 will be lost when this function is
        called.

    INPUTS
        Netbuff0        Pointer to source NetBuff structure.
        Netbuff1        Pointer to destination NetBuff structure.

    RESULTS
        error           Zero if successful; non-zero otherwise.

    NOTES
        This function may be called from interrupts.

        Only the first NetBuffSegment in NetBuff1 is guaranteed to
        still be in NetBuff1 when this function returns.

    SEE ALSO
        netbuff.library/CopyFromNetBuff(),
        netbuff.library/CopyToNetBuff()

    BUGS
```

```
netbuff.library/CopyToNetBuff()

   NAME
        CopyToNetBuff -- Replace data in a NetBuff.

   SYNOPSIS
        error = CopyToNetBuff( Netbuff, Offset, Data, Count )
        D0                     A0       D0      A1    D1

        LONG CopyToNetBuff( struct NetBuff *, LONG, UBYTE *, ULONG ):

   FUNCTION
        This function replaces existing data in Netbuff with 'Count'
        bytes from those pointed to by Data. If Offset is
        non-negative, then the replacement starts 'Offset' bytes from
        the start of Netbuff. If Offset is negative, then the
        replacement starts 'abs(Offset)' from the end of Netbuff.

        This function will not change the amount of data that Netbuff
        contains; it will only replace parts of it.

   INPUTS
        Netbuff         Pointer to NetBuff structure to copy data to.
        Offset          Offset into NetBuff to place data.
        Data            Pointer to data to copy.
        Count           Number of bytes of data to copy.

   RESULTS
        error           Zero if successful; non-zero otherwise.

   NOTES
        This function may be called from interrupts.

        This function might be used within a network device driver to
        fill a NetBuff from bytes taken from the hardware. In this
        case, CopyToNetBuff()s would be preceded possibly by a call to
        ReadyNetBuff().

   SEE ALSO
        netbuff.library/CopyFromNetBuff(),
        netbuff.library/ReadyNetBuff()

   BUGS
```

```
netbuff.library/FreeSegments()

   NAME
        FreeSegments -- Return a list of NetBuffSegments.

   SYNOPSIS
        FreeSegments( Segment_list )
                      A0

        void FreeSegments( struct List * );

   FUNCTION
        This function gives a list of NetBuffSegments to the system
        free pool.

   INPUTS
        Segment_list    Pointer to the list of NetBuffSegments to add
                             to the system free NetBuffSegment pool.

   NOTES
        When this routine encounters a NetBuffSegment with
        PhysicalSize of 0, the data area is left untouched but the
        NetBuffSegment structure which points to the data is freed
        using exec.library/FreeMem().

   SEE ALSO
        netbuff.library/IntFreeSegments(),
        netbuff.library/AllocSegments(),
        netbuff.library/ReadyNetBuff()

   BUGS
```

```
netbuff.library/IntAllocSegments()

    NAME
        IntAllocSegments -- Get a list of NetBuffSegments.

    SYNOPSIS
        IntAllocSegments( Count, Segment_List )
                          D0     A0

        void IntAllocSegments( ULONG, struct List * );

    FUNCTION
        This function returns a list of NetBuffSegments sufficient to
        hold Count bytes.

    INPUTS
        Count           Numbers of bytes for which space is needed.
        Segment_List    Pointer to an empty (initialized) list to hold
                            the allocated NetBuffSegments.

    RESULTS

    NOTES
        The function should be called only from interrupts.

    SEE ALSO
        netbuff.library/AllocSegments(),
        netbuff.library/IntFreeSegments(),
        netbuff.library/ReadyNetBuff()

    BUGS
        Since this function may be called from interrupts, it can not
        actually allocate memory from the system. It, therefor, relies
        on a "friendly" task or process to add NetBuffSegments to the
        free pool via the netbuff.library/FreeSegments() function.
```

```
netbuff.library/IntFreeSegments()

    NAME
        IntFreeSegments -- Return a list of NetBuffSegments.

    SYNOPSIS
        IntFreeSegments( Segment_list )
                         A0

        void IntFreeSegments( struct List * );

    FUNCTION
        This function gives a list of NetBuffSegments to the system
        free pool.

    INPUTS
        Segment_list    Pointer to the list of NetBuffSegments to add
                            to the system free NetBuffSegment pool.

    NOTES
        The function should be called only from interrupts.

        When this routine encounters a NetBuffSegment with
        PhysicalSize of 0, that data area is left untouched but the
        NetBuffSegment structure which points to the data is freed
        using exec.library/FreeMem().

    SEE ALSO
        netbuff.library/FreeSegments(),
        netbuff.library/IntAllocSegments(),
        netbuff.library/ReadyNetBuff()

    BUGS
        This function relies on the non-interrupt functions to perform
        garbage collection of segments of physical size zero.
```

```
netbuff.library/IsContiguous()

    NAME
        IsContiguous -- Checks if data in in contiguous memory.

    SYNOPSIS
        result = IsContiguous( Netbuff, Offset, Count )
        D0                     A0       D0      D1

        LONG IsContiguous( struct NetBuff *, LONG, ULONG );

    FUNCTION
        This function indicates whether or not Count bytes of data
        starting at Offset in NetBuff are in contiguous bytes. If
        Offset is non-negative, then the start of the data to check is
        'Offset' bytes from the start of the NetBuff. If Offset is
        negative, then the start of the data to check is 'abs(Offset)'
        bytes from the end of the NetBuff.

    INPUTS
        Netbuff         Pointer to NetBuff to check.
        Offset          Offset into NetBuff where to check.
        Count           Number of bytes that should be contiguous.

    RESULTS
        result          Zero if non-contiguous; non-zero otherwise.

    NOTES

    SEE ALSO

    BUGS
```

```
netbuff.library/NetBuffAppend()

    NAME
        NetBuffAppend -- Append one NetBuff to then end of another.

    SYNOPSIS
        error = NetBuffAppend( Netbuff0, Netbuff1 )
        D0                     A0        A1

        LONG NetBuffAppend( struct NetBuff *, struct NetBuff * );

    FUNCTION
        This function appends the contents of Netbuff1 to the end of
        Netbuff0.

    INPUTS
        Netbuff0        NetBuff to be appended to.
        Netbuff1        NetBuff to append.

    RESULTS
        error           Zero if successful; non-zero otherwise.

    NOTES

    SEE ALSO
        netbuff.library/PrependNetBuff(),
        netbuff.library/SplitNetBuff()

    BUGS
```

```
netbuff.library/PrependNetBuff()

   NAME
        PrependNetBuff -- Prepend one NetBuff to the front of another.

   SYNOPSIS
        error = PrependNetBuff( Netbuff0, Netbuff1 )
        D0                      A0        A1

        LONG PrependNetBuff( struct NetBuff *, struct NetBuff * );

   FUNCTION
        This function prepends the contents of Netbuff1 to the front
        of Netbuff0.

   INPUTS
        Netbuff0        NetBuff to be prepended to.
        Netbuff1        NetBuff to prepend.

   RESULTS
        error           Zero if successful; non-zero otherwise.

   NOTES

   SEE ALSO
        netbuff.library/NetBuffAppend(),
        netbuff.library/SplitNetBuff()

   BUGS
```

```
netbuff.library/ReadyNetBuff()

   NAME
        ReadyNetBuff -- Ready a NetBuff for copying to.

   SYNOPSIS
        error = ReadyNetBuff( Netbuff, Count )
        D0                    A0       D0

        LONG ReadyNetBuff( struct NetBuff *, ULONG );

   FUNCTION
        This function sets the amount of data refered to by a NetBuff,
        and the associated NetBuffSegments, to Count bytes in
        preparation for a CopyToNetBuff() operation. It does not
        initialize the data in the NetBuff.

   INPUTS
        Netbuff         Pointer to NetBuff structure to initialize.
        Count           The number of bytes of data that the NetBuff
                            is to refer to.

   RESULTS
        error           Zero if successful; non-zero otherwise.

   NOTES
        This function may be called from interrupts.

        This function will attempt to allocate NetBuffSegments as
        needed to make room for Count bytes available.

        Unneeded NetBuffSegments will be returned to the system free
        pool.

   SEE ALSO

   BUGS
```

```
netbuff.library/ReclaimSegments()

  NAME
       ReclaimSegments -- Free all but first NetBuffSegment.

  SYNOPSIS
       error = ReclaimSegments( Netbuff )
       D0                       A0

       LONG ReclaimSegments( struct NetBuff * );

  FUNCTION
       This function frees all non-zero PhysicalSize NetBuffSegments
       except the first NetBuffSegment of the NetBuff.

  INPUTS
       Netbuff          Pointer to NetBuff to reclaim segments from.

  RESULTS
       error            Zero if successful; non-zero otherwise.

  NOTES
       This function may be called from interrupts.

       The intended use for this function is to encourage drivers or
       protocol stacks to free NetBuffSegments from NetBuffs in read
       queues.

  SEE ALSO

  BUGS
```

```
netbuff.library/SplitNetBuff()

  NAME
       SplitNetBuff -- Split a NetBuff in to two NetBuffs.

  SYNOPSIS
       error = SplitNetBuff( Netbuff0, Count, Netbuff1 )
       D0                    A0        D0     A1

       LONG SplitNetBuff( struct NetBuff *, LONG, struct NetBuff * );

  FUNCTION
       This function takes a NetBuff and splits it at the specified
       offset. The sign of Count determines which bytes will be
       removed from Netbuff0 and placed in Netbuff1. If Count is
       non-negative, the first 'Offset' bytes in Netbuff0 will be
       moved to Netbuff1. If Offset is negative, the last
       'abs(Offset)' bytes in Netbuff0 will be moved to Netbuff1.

       All data in Netbuff1 will be lost when this function is
       called.

  INPUTS
       Netbuff0         Pointer to NetBuff structure with data to
                            split.
       Count            Number of bytes to split off.
       Netbuff1         Pointer to a NetBuff structure to receive
                            split-off data.

  RESULTS
       error            Zero if successful; non-zero otherwise.

  NOTES
       This function might be used within a protocol stack to split
       a packet into smaller NetBuffs so as to fall below a specific
       transport medium's maximum transfer unit.

  SEE ALSO
       netbuff.library/TrimNetBuff(),
       netbuff.library/NetBuffAppend(),
       netbuff.library/PrependNetBuff()

  BUGS
       If the split point is in a NetBuffSegment of PhysicalSize
       zero, exec/AllocMem() will be called to create a new segment
       of PhysicalSize zero.
```

```
netbuff.library/TrimNetBuff()

   NAME
        TrimNetBuff -- Elliminate leading or trailing data.

   SYNOPSIS
        error = TrimNetBuff( Netbuff, Count )
        D0                   A0       D0

        LONG TrimNetBuff( struct NetBuff *, LONG );

   FUNCTION
        This function takes a NetBuff and elliminates 'abs(Count)'
        bytes of data from the first or last bytes of the NetBuff data
        depending on the sign of Count. If Count is positive, the
        first 'Count' bytes of data are elliminated. If Count is
        negative, the last 'abs(Count)' bytes are elliminated.

        Emptied NetBuffSegments (which may result) will be returned to
        the free pool.

   INPUTS
        Netbuff         Pointer to NetBuff to be trimmed.
        Count           Number of data bytes to remove.

   RESULTS
        error           Zero if successful; non-zero otherwise.

   NOTES
        This function might be used within a protocol stack to remove
        levels of protocol wrapping from either side of a packet.

   SEE ALSO
        netbuff.library/SplitNetBuff(),
        netbuff.library/NetBuffAppend(),
        netbuff.library/PrependNetBuff()

   BUGS


   CREDITS
        Raymond S. Brand,   rsbx@cbmvax.commodore.com,  (215) 431-9100
        Martin Hunt,      martin@cbmvax.commodore.com,  (215) 431-9100
        Perry Kivolowitz,            ASDG Incorporated,  (608) 273-6585
        Dale Luck,                     dale@boing.uucp,  (408) 262-1469
```

Scalable Outline Fonts

Bob Burns - Kodiak Software

A Year of Implementation

At the 1990 developers conferences, Commodore presented ''Scalable Fonts: A Decade of Change''. It consisted of an overview of outline font scaling in general, and mention of Commodore's acquisition of AGFA Compugraphic's ''Intellifont'' outline font engine.

Scalable outline fonts are now a part of the Amiga OS. Commodore has ported AGFA Compugraphic's ''Bullet'' outline font engine--so named because it is a faster version of their ''Intellifont'' code. Commodore's port occurred in two phases. The first phase resulted in the version 37 *diskfont.library* on the 2.04 release disks. The second phase resulted in the version 38 *diskfont.library* and *bullet.library* on the 2.1 pre-release disks. Both provide 1.3-compatible applications access to Amiga fonts created on the fly from outlines. Both extend the existing diskfont interface to allow new applications to better define the characteristics of a font. The second phase also provides direct access to the outline engine.

Effortless Outline Fonts

When you install release 2.04 on your hard disk, the version 37 *diskfont.library* will be installed into your LIBS: directory, and three outline typefaces will be installed into your FONTS: directory. These three typefaces consist of one fixed width, one serif, and one sans-serif typeface, called LetterGothic, CGTimes, and CGTriumvirate, respectively. When you invoke tools that use Amiga fonts, you will find these three new fonts available in sizes 15, 30, 45, 60, and 75 for your use.

Though the differences between these outline fonts and traditional Amiga bitmap fonts should be invisible to the application, you may observe some differences. When you first access a specific size, you will notice that it takes longer than usual to load the font in from disk. This is because the bitmap for the font is not what is being read off the disk--the bitmap is actually created by the *diskfont.library* from the outline description. If your application allows the user to specify an arbitrary font size, the new size will be created fresh from the outline, not scaled from one of the specified sizes. If your application asks for bold and/or italic typefaces in the OpenDiskFont() call before falling back on the *graphics.library's* algorithmic bolding or italicizing, you will notice the improved bold and italic transformations that the scaling engine provides.

On my A3000, I take advantage of the fact that, under 2.0, the FONTS: directory can be an assign path. I leave the Commodore Workbench image on the System2.0: partition relatively untouched and I assign FONTS: to both *System2.0:Fonts/* and *Work:Fonts/*, and I install any other fonts I get into *Work:Fonts/*. This strategy works fine for outline fonts, too as they use the FONTS: assign path appropriately. An extension of this strategy that some may find helpful would be to keep groups of typefaces in separate directories (e.g. publishing faces, slide show faces, outline faces, ...), and assign FONTS: to whatever aggregate of groups is appropriate before launching an application. This allows folks with a large number of fonts to avoid confusing applications that can only handle a limited number of fonts. Applications that want to open their own fonts explicitly can specify the font path, e.g. OpenDiskFont({"Phontom:fonts/crawler.font", 12, 0, 0}). Outline fonts may also be explicitly specified in this manner.

Note that the *diskfont.library* on the actual 2.04 Workbench2.0 floppy is a *version 36 diskfont.library* with none of the outline font handling features. The *version 37 diskfont.library* is installed onto your hard disk from the disk AmigaFonts2.0. This is because there are times when the outline font support is not desired: it is very slow when used with floppies, and it is a significantly larger library (50K versus 8K). Thus, release 2.04 is structured so that users that run off floppy disks by cloning the Workbench disk (typically vanilla A500 users) will be using version 36, and users that install the workbench onto a hard disk (hopefully with systems with more RAM) will be using version 37. With release 2.1, the bullet code has been taken out of the *diskfont.library*, bringing the size of the *diskfont.library* back down (though only to 14K). The *bullet.library* is only invoked by version 38 *diskfont.library* when required. At press time, it is expected that release 2.1 will contain only the version 38 *diskfont.library*.

Fount of Knowledge

The description of a bullet outline typeface is contained in a set of three related files. The installation and manipulation of these files is performed via the *Fountain* utility. Outline typefaces don't intrinsically have sizes associated with them to return to applications as "available" (i.e. as a result of the AvailFonts() function). The sizes 15, 30, 45, 60, and 75 are returned for the three typefaces on AmigaFonts2.0 because these were the sizes specified when they were installed using *Fountain*. *Fountain* lets the user both install new outline typefaces and modify existing ones.

Outline typefaces come from one of two sources: AGFA Compugraphic's FAIS disks (which are published on MS-DOS compatible disks) and existing outline files. When installing fonts from a FAIS disk, Fountain generates the outline file. The outline font disks from Gold Disk already have the outline file on the disk. Whenever you install a typeface, the fountain environment variable ENV:Sys/*Fountain* is used to initialize the list of sizes that will typically be presented by applications in their font menus. If no such environment exists, it defaults to 15, 30, 45, 60, and 75. The smaller sizes would be appropriate for screen display of document text, the larger sizes for presentation graphics. These sizes can be changed for an individual typeface with the *Fountain* ''Modify Existing Typefaces...'' screen, but if you find yourself always changing to a different list, you should change this environment variable. The format of the environment variable is a file with a list of sizes in text (ASCII) form. There can be no more than 20 sizes in this list.

Fountain also facilitates the creation of bitmap images of specific sizes from a font outline, which can be saved to disk using the traditional font file format. This feature would be used to pre-create sizes that are often used, so that they may be loaded into memory faster when opened.

More Effort, More Effect

There are new 2.0 interfaces to the *diskfont.library* that an application can use to get more control over the creation of fonts. These are accessed by passing the new tagged text attribute structure TTextAttr to OpenDiskFont(). The flag `FSF_TAGGED` is set in the ta_Style byte to indicate that the text attribute is a TTextAttr structure and not a TextAttr structure--i.e. that the new field tta_Tags field exists at the end of the structure. These structures are defined in the include file *<graphics/text.h>*. The tag list can contain entries that describe the device resolution and dot size, or specify a size by point size instead of Y height. In version 37, the symbol set can be specified, but this concept is obsolete in version 38 (this will be discussed later). These tags are described in *<libraries/diskfonttag.h>*.

Specification of a device resolution by the application can be used to distort the aspect ratio of a font from the default 1:1 ratio. For example, if an application knows that fonts will be used in the Hires display mode, it can adjust the aspect ratio appropriately by changing the DPI values to 100 50 by supplying a OT_DeviceDPI tag with the value 0x00640032.

Note that significantly more tags are defined in this file than are valid for use in a tta_Tags list: only OT_DeviceDPI, OT_DotSize, OT_PointSize, and in version 37, OT_SymSet, and in version 38, OT_PointHeight will have effect in tta_Tags.

Whenever the *diskfont.library* creates an outline typeface description into an Amiga graphics font, the diskfont environment variable ENV:Sys/Diskfont is used to specify parameters that are not supplied via a tag list by the application. The template for the environment variable is XDPI/N, YDPI/N, XDOTP/N, YDOTP/N, and in version 37, SYMSET. The diskfont environment is only read by the *diskfont.library* once, at boot time. This limits the usefulness of this environment--the default aspect ratio can be distorted to be proper for only one display mode.

Fountain also looks at the diskfont environment when it is invoked. Note that if the environment is changed after boot but before invoking *Fountain*, *Fountain* will have a different environment than the *diskfont.library*. Though this can be used to create Amiga bitmap font images with different aspect ratios, the distorted bitmaps also contain their aspect information, and thus will not be used by the *diskfont.library* if that aspect ratio does not match the diskfont environment. The *diskfont.library* will regenerate a bitmap image from the outline that *does* match the environment. In other words, the *diskfont.library* will enforce the aspect ratio specified in its environment. The only way for users to have fonts with multiple aspect ratios available to use with 1.3 applications is to rename these distorted bitmaps into some other font name directory, invoke *FixFonts*, and treat them as regular Amiga bitmap fonts.

Remember also that when a stylistic variation of a typeface is desired, there is now a reason to set that style (bold or italic) in the ta_Style flags and try to OpenDiskFont() it before applying the SetSoftStyle() function. The results of the outline engine creating these variations on the outline is much better than the results of the blitter creating them on the bitmap.

Bite the Bullet

Release 2.1 separates the bullet font rendering code back out of the *diskfont.library* (where it is in 2.04) and makes it a separate Amiga Exec library. The function interface to this library is very simple. There are five basic functions that are complete and sufficient to access the bullet code. OpenEngine() and CloseEngine() provide and release the engine handle, which is a pointer to a data area that the library uses to store the setup for the caller. The first part of this handle is public. It is the GlyphEngine structure defined in *<libraries/glyph.h>*. SetInfo() sets attributes of the typeface, and ObtainInfo() and ReleaseInfo() manage both typeface inquiry and the results of any rendering operations. The calls are tag based, and are thus extensible. There is also one shortcut function, GetGlyph(), that combines the effect of several XxxxInfo() calls.

The two functions pairs OpenEngine() and CloseEngine(), and ObtainInfo() and ReleaseInfo() provide the extension to the Amiga Exec library interface necessary to handle the scoping of data i.e. it provides the definite beginning and ending markers for an application's use of bullet data. The engine pair tells the library when a specific application wants to use *bullet.library*, and provides a data area to be used by the library exclusively on behalf of that application. The info pair tells the library when an application needs data from *bullet.library*, and when the application is done with that data and the associated storage can be reclaimed. All the *bullet.library* functions are more completely described in Appendix A, which is the *bullet.library* Autodoc.

The use of tags provides application control over all aspects of the bullet engine, allowing the specification of the size, aspect ratio, shear (pseudo-italicizing), rotation, and emboldening of individual character bitmaps. Moreover, it allows inquiry of both nominal character spacing and of character pair kern adjustments. The tags are described in Appendix B, which contains the version 38 *<libraries/diskfonttag.h>* and *<libraries/glyph.h>* files.

Bear-Brained Ideas

The design of the Amiga interface for outline font support was guided by two principles: first, to make outline fonts seamlessly available to existing applications, and second to make the new interface to outline fonts extensible, robust, and vendor independent.

The support of outline fonts via the *diskfont.library* achieves the first purpose. Applications that need no more than traditional Amiga fonts find that their use of the *diskfont.library* to access outline fonts occurs without any extra effort on their part. The primary benefits of the use of outline fonts--access to professional typefaces at arbitrary sizes--are provided via this interface. This is the only interface available in release 2.04.

Applications that require more from the outline font engine--more sophisticated transformations such as shear and rotation, more precise character placement information, and characters outside the Amiga 8-bit character set--will use the *bullet.library* interface. This interface is tag based and provides explicit mechanisms to define the duration of access to data. This provides straight forward extensibility. It is not a direct port of the AGFA Compugraphic Bullet functions as there is nothing in the interface that would not be equally valid as an interface for a different vendor's outline font engine. Moreover, the interface from the *diskfont.library* to the *bullet.library* is just as flexible. The string "bullet.library" does not even exist in the *diskfont.library*. Access from the *diskfont.library* to the *bullet.library* is defined by a typeface's outline tag file, which is one of the three files created by *Fountain* during the installation process. If a different vendor's font engine were created, the *diskfont.library* would not need to be updated to use it. This is why there is no *<libraries/bullet.h>* file. The structures that are used to access various elements of *bullet.library* data are found in *<libraries/glyph.h>*, and are called GlyphEngine and GlyphMap, not BulletEngine and BulletMap.

Inside the Cave

The installation of a bullet typeface is a transformation by *Fountain* from one of two types of files into the three installation files. The three installation files created by *Fountain* are the *.font* file, the *.otag* file, and the *.type* file. The *.font* file is compatible with the format of existing *.font* files, with a different identifier that tells the *diskfont.library* to go look for the *.otag* file. The *.otag* file lives in the same directory as the *.font* file, and consists of an image of a tag list that describes the typeface. This tag list contains tags as described in Appendix B. The *.type* file contains the actual bullet-specific typeface description, and lives in the *_Bullet_Outlines/* directory, which is a sibling of the *.font* file.

The *.otag* file contains information useful to applications. It describes the typeface style, weight, horizontal style, slant, whether it is fixed width or proportional, and other typefaces in the same family. For example, the CGTimes typeface will contain references to CGTimesBold, CGTimesItalic, and CGTimesBoldItalic as family members to use to achieve bold, italic, and bold italic variations if these family members were also installed by *Fountain*. The *.otag* file also contains the name "bullet" and the bullet-specific data required by the bullet outline engine to access the typeface. The *.otag* file is the disk file form of a memory taglist image. When this taglist is moved into memory it can contain pointers. Those tags with pointer data have tag values with the `OT_Indirect` bit set. In the *.otag* file, the pointer tag's data value contain the byte offset from the beginning of the file to the data the tag points to.

The two different types of files on the input side of *Fountain* are FAIS files and outline files. FAIS is an AGFA Compugraphic acronym for their *Font Access and Interchange Standard.* FAIS typefaces are commercially available on MS-DOS disks. They are usually packaged as four or eight related typefaces to a disk. They can be read on an Amiga into *Fountain* via *CrossDos* or an equivalent. Outline files are simply *.type* files that have already been processed for use on a 680x0 implementation of the bullet engine. Gold Disk sells AGFA Compugraphic typefaces in this form for use with their products. *Fountain* produces these files.

A *.type* file is a reordering of the useful parts of an FAIS file to make it suitable for use by the outline font code. If you are interested in editing outline fonts, edit FAIS-format files and let the *Fountain* application make the *.type* file for you from the FAIS file. This will make your application more portable as FAIS files are understood not only on the Amiga but also by tools on other platforms, including those that download them to the HP LaserJet III (which uses the AGFA Compugraphic code internally). The format of FAIS files is described in a 2cm thick document. This document is published by AGFA Compugraphic, and is only available directly from them. Tim Christo has bravely suggested that his name be mentioned as a contact person to acquire this document:

Tim Christodoulopoulos
Project Manager/ISV Technical Support
90 Industrial Way, MS 90-1-3C
Wilmington, MA 01887
USA
Telephone (508) 658-0200 extension 2069

There is a collection of files required by the bullet code itself that is installed in the *Fonts:_Bullet/* directory as part of the workbench installation process. They are *if.fnt*, *plugin.type*, and for version 37 *if.ss*, and for version 38 *if.uc*. The file *plugin.type* contains the outline descriptions for those characters that do not vary from typeface to typeface, or only vary depending on whether the typeface is serif or sans-serif, and normal or italic. An example of a character that never varies is the bullet symbol. An examples of a character that has limited variation is a bracket. The file *if.fnt* is an index into *plugin.type*. The files *if.ss* and *if.uc* describe the mapping from Amiga character codes to AGFA Compugraphic internal character codes.

Caveats

There are some aspects of the outline font implementation that are not unanimously acclaimed as "features".

Fonts created from outline descriptions by the *diskfont.library* do not have the `FPB_DESIGNED` bit set. They do have the `FPB_DISKFONT` bit set. This yields:

(Topaz 8 & 9) from ROM:	`ROMFONT | DESIGNED`
Outline fonts:	`DISKFONT`
Any font directly from disk:	`DISKFONT`
Bitmap fonts from disk:	`DISKFONT |` (usually) `DESIGNED`
Any bitmap scaled font:	<none>

So, any font with either the `FPB_ROMFONT` or `FPB_DISKFONT` flag set can be construed not to have been created by bitmap scaling of another font. Though it is always true that a font with the `DESIGNED` bit is a bitmap font, the *diskfont.library* does not enforce that it exists for a bitmap font, so the converse is not always true--there is no way to tell whether a font with the `DISKFONT` but not the `DESIGNED` bit was generated from outline or was loaded from a font that does not have the `DESIGNED` bit set. Had I a chance to do things again, I would force the `DESIGNED` bit for disk bitmaps. Further, I would reflect this change in the AvailFonts() entry and perhaps set the `AFF_BITMAP` bit for bitmap entries, and not set it for outline entries. But as things stand, the only way to determine if a face name is associated with an outline typeface is to look at the contents of the associated *.font* file itself.

When the characters comprising the bitmap images of an outline font are composed into an Amiga font, special code in the *diskfont.library* makes sure that there is sufficient whitespace between the characters for small sizes for legibility. Additionally, the Intellifont algorithm used by the outline font engine tries to keep stem widths of characters consistent, and there is thus a noticeable jump in the weight of characters at small sizes as the stem widths jump from 1 to 2 to 3. These two effects result in a non-linear relationship between the height of a font and its width for small sizes. There will even be times when the text for a taller size is less wide than for a shorter size. The *diskfont.library* code emphasizes the screen legibility of the font. Some applications might prefer that the uniform scaling of a font be emphasized from the screen to an ideal output device. When an application interfaces to the *bullet.library* itself to get character bitmaps, it gets all the information necessary to position the character as close to an ideal position as possible.

There are three problems with aspect ratio distortion. The first problem is that the *graphics.library* WeighTAMatch() in versions 36 and 37 has a bug in it that causes differences in aspect ratios to be ignored if the height is OK. The result of this is that once a font is created at a particular height, no image of the same height with a different aspect will be created (The patch program *SetPatchWTAM* will fix this flaw). The second problem is that if a distorted font is asked for, and then an application that *doesn't* specify the aspect asks for a font at the same height, that application will get the distorted font (This is a bug). The third problem is that the OT_DeviceDPI tag is effective in distorting the aspect ratio only for fonts that have information about their own aspect. Most bitmap fonts stored on disk do not have this information (none are known at press time to ship with this information). If the aspect for a source font is not known at a particular height, no distortion of that aspect is performed.

This means that it is not possible to create a double-wide topaz 8. Paradoxically, it *is* possible to create a double-wide topaz 10. This is because the ROM image for topaz 9 *does* have aspect information in it, thus since topaz 10 does not exist, it can be distorted during scaling. Moreover, the scaled result will have aspect information stored in it.

There is nothing to prevent a program from asking for a typeface that is invalid because it is too large, nor is there a way to inquire what this size is. As a good rule of thumb, sizes less than 128 will have no trouble as long as there is enough memory, and sizes greater than 256 are increasingly likely to be too big. This is because when the total black width of a typeface exceeds 2^16 pixels, the character pointers within the font can no longer point to the start position of the character image, thus the font is corrupt.

The internal character codes used by AGFA Compugraphic may vary according to the FAIS character complement i.e. according to a code that AGFA uses to indicate the purpose of the collection of the characters in the typeface. Two unfortunate things have occurred: first, there are several character complements that Amiga users will encounter, and they don't always have exactly the same character codes for specific characters, and second, these character complement numbers are stripped from the typeface in the process that converts it to a *.type* file. In version 37 the bullet executable is internal to the *diskfont.library*, and the mapping file *if.ss* controls the mapping of Amiga's 8-bit character set to the internal AGFA codes. The multiple mapping possibilities are described by different symbol sets within *if.ss*. There are two symbol sets that are currently present: "L1" and "GD". The first, L1, is used with all FAIS disks distributed by AGFA Compugraphic and with the typefaces that come on the Workbench disk. The second, GD, is used with typefaces distributed by Gold Disk. Since the character complement information is not available in the typefaces distributed by Gold Disk, these symbol sets must be specified by the user. Thus, when installing (or modifying) typefaces from Gold Disk under the version 37 *diskfont.library*, the user must set an *Env:sys/diskfont* that contains the line "SYMSET GD". This will be saved by *Fountain* into the *.otag* file, and this setting will thus be known whenever the associated typeface is referenced. The SYMSET environment variable must be reset to L1 whenever an FAIS or Workbench outline font is installed (or modified).

One World, One Code

For version 38, two factors made retaining the symbol set metaphor untenable: first the symbol set metaphor was a kludge specifically for AGFA Compugraphic. SYMSET was being used to fix the problem generated by the multiple AGFA Compugraphic typeface maps not to distinguish between different character uses. Most of the L1 and GD symbol sets were the same. A scheme that used a primary mapping and then fell back to alternate mappings if the primary map did not exist was found to work, and was implemented (though not in time to change version 37).

Second, direct access to the outline engine freed the interface to allow access to characters besides the Amiga's 192 characters. Since the typefaces associated with the bullet engine did in fact describe hundreds of additional characters, it seemed appropriate to devise a code to access them. The work of several standards organizations for the extension of 8-bit characters was investigated, and the Unicode model was deemed most suitable and adopted. At press time, the resulting character map had not yet been finalized. This map consists of the intersection of the characters supplied by the current implementation of the *bullet.library* with those described in the current draft of Unicode. It is this map that is described in the version 38 file *if.uc*.

Note that Unicode provides character codes for many more characters. If and when characters are added to the set described above, they would need to be given AGFA Compugraphic character numbers and the file *if.uc* would need to be expanded. If the character numbers were not unique, the user would be responsible for ensuring that the correct typeface were used -- for example, that the Cyrillic typeface were specified before asking for Cyrillic characters (which, in AGFA Compugraphic's current numbering scheme, are given internal character numbers that are the same as the Latin characters). I hope instead that the internal character numbers are made unique, either at the FAIS source, or by another Amiga installation utility.

More Bits

The following programs are on the conference disks. They were developed to run on an A3000 in interlace. They are not polished, but they contain useful code fragments:

SetPatchWTAM - This SetPatch-like program patches *WeighTAMatch* in the V37 *graphics.library* to correctly report fonts with the same height but different aspect ratios as different.

SOFDemo - This program shows how to open an outline font with the appropriate style and aspect ratio. Use *SetPatchWTAM* before executing.

Height - This old test program displays character samples for lots of font sizes. Note the way characters are sometimes shifted to ensure they fit into the smaller Y sizes.

PrtFace - This old test program prints a font sampler using the *bullet.library*. Until the map is finalized, do not trust the codes beyond hex FF.

Spacing - This old test program displays sample lines for the font sizes that are affected by the special code that the *diskfont.library* uses to ensure small characters don't run together.

Spiral - This old test program demonstrates direct access to the *bullet.library* to obtain arbitrarily sized and rotated character glyphs. It's *almost* a spiral, but not quite.

Strikefont - This old test program displays all the characters in a typeface by directly accessing the *bullet.library*.

Test - This old test program tests various aspects of the bullet engine and reports the results via printf()s. Use the numeric parameters to change default values and the switch parameters to turn on various outputs.

Appendix A

bullet.library/--background--

AMIGA FONTS

The existance of a disk font is indicated by the existance of its associated font contents file, whose name has the suffix ".font". It is this name that is used in the actual font open call of the application. Amiga fonts are collected in the directory path(s) associated with the FONTS: assign. It is this assign that is searched if no explicit path name is provided in the font open call: use of an explicit path is generally discouraged. The actual bitmaps of traditional Amiga fonts are stored in a directory with the name of the font contents file stripped of its ".font" suffix. This directory is usually in the same directory as the font contents file. This traditional arrangement is supported by the FixFonts system application.

For example:

- o FONTS: is assigned to Sys:Fonts/
- o Sys:Fonts/garnet.font exists and describes that the font "garnet.font" exists
- o Sys:Fonts/garnet/ contains the bitmap images for sizes 9 and 16

Other variations of file placement may exist, but they require custom tools to maintain -- tools available not from Commodore, but from other sources such as Fish disks.

Font contents files are flagged with magic numbers that not only verify that they are a font contents files but also what variation of file structure they contain.

OUTLINE TYPEFACES

The existance of an outline typeface is indicated by a magic number in the font contents file. They are further described in the associated outline typeface tag file, whose name is the that of the font contents file with the suffix ".otag" substituted for ".font". This tag file contains a tag list that is to be processed and passed to the outline engine (i.e. bullet.library) in order to select the associated typeface. It also contains information applications may use to guide their use of the typeface.

OTAG SPECIFICATION EXAMPLE

Here are the steps necessary to go from an arbitrary font name into an environment where glyphs from that font can be accessed:

1. Read the header from the font contents (.font) file and verify that the magic cookie fch_ID is OTAG_ID. If it is not, then this is an Amiga bitmap font, not an outline font.
2. Read the associated outline tag (.otag) file into memory:
   a. Validate that the OT_FileIdent exists and matches the file size.
   b. Allocate a memory buffer and read the file into it.
   c. Resolve addresses: for each tag with the OT_Indirect bit set, add the memory buffer origin to the associated data.
3. Find the OT_Engine tag and ensure that you have the proper engine open:
   a. If you already have an engine handle for this engine name, you skip these steps.
   b. append the suffix ".library" to the engine name. (e.g. "bullet" becomes "bullet.library").
   c. use exec's OpenLibrary() to open the library.
   d. use the engine's OpenEngine() to acquire an engine handle.
4. Pass the full path name of the .otag file to the engine with the OT_OTagPath tag using SetInfo().
5. Pass the memory copy of the .otag file to the engine with the OT_OTagList tag using SetInfo(). This step may be combined with step 4, passing first the path then the list in one call.

The library is now ready to accept glyph metric information (e.g. size and glyph code) and produce glyph bitmaps.

DISKFONT USE OF OTAG ENTRIES

The diskfont library uses other entries from the outline tag (.otag) file. The following are used both during inquiry of what typefaces exist (AvailFonts) and during creation of an Amiga TextFont (OpenDiskFont):

- o OT_IsFixed is used to determine whether these outlines describe a PROPORTIONAL flagged font.
- o OT_StemWeight is used to determine whether these outlines describe a BOLD style font.
- o OT_SlantStyle is used to determine whether these outlines describe an ITALIC style font.
- o OT_HorizStyle is used to determine whether these outlines describe an EXTENDED style font.

The following are used only during OpenDiskFont:

- o OT_YSizeFactor is used to convert the Amiga pixel height specification, which describes the distance from the lowest decender to the highest ascender, into a point size specification, which is related (via YSizeFactor) to a nominal character height.
- o OT_SpaceWidth is used as the width of the space character.

The following is used only during AvailFonts:

- o OT_AvailSizes is used to generate a list of sizes available for the font.

```
bullet.library/CloseEngine

    NAME
      CloseEngine -- Release an engine handle

    SYNOPSIS
      CloseEngine(engineHandle)
                  A0

      void CloseEngine(struct GlyphEngine *);

    FUNCTION
      This function releases the engine handle acquired with
      OpenEngine.  It first releases any data acquired with
      ObtainInfoA associated with the engineHandle that has not yet
      been released.

    INPUTS
      engineHandle -- the handle acquired via OpenEngine.  If zero,
            no operation is performed.

    RESULT
      This function has no result.  The only error that can occur is
      when the when an invalid engineHandle is supplied: the
      application is assumed not to do that.

    EXAMPLE
      EndGame(code, arg1, arg2, arg3, arg3)
      {
          ...
          CloseEngine(EngineHandle);
          ...
      }

    SEE ALSO
      OpenEngine()
```

```
bullet.library/GetGlyphMap

    NAME
      GetGlyphMap -- Acquire a glyph bitmap.

    SYNOPSIS
      error = GetGlyphMap(engineHandle, glyphCode, glyph)
                          A0            D0         A1

      ULONG GetGlyphMap(struct GlyphEngine *, int, struct Glyph **);

    FUNCTION
      This function provides a shortcut way to access a glyph
      bitmap.  It does the equivalant of the following function
      sequence:
      1.  If a previous GetGlyphMap has been performed with this
          engine handle: ReleaseInfoA the previous glyph.
      2.  SetInfoA the OT_GlyphCode to glyphCode.
      2.  ObtainInfoA the OT_Glyph into this glyph.

    INPUTS
      engineHandle -- the handle acquired via OpenEngine.
      glyphCode -- the glyph code whose image is to be acquired.
      glyph -- a pointer to store the pointer to the Glyph structure
            that contains the glyph metrics and bitmap.

    RESULT
      This function returns a zero success indication, or a non-zero
      error code.

    EXAMPLE
          struct Glyph *glyph;

          for (code = 32; code <= 255; code++) {
            if (code == 128)
                code = 160;
            if (!GetGlyphMap(EngineHandle, 0x32323200 | code, &glyph)) {
                ...
            }
          }

    SEE ALSO
      ObtainInfoA(), ReleaseInfoA(), SetInfoA(), libraries/oterrors.h
```

```
bullet.library/ObtainInfoA

  NAME
    ObtainInfoA -- Inquire tagged font and/or glyph metrics
    ObtainInfo -- varargs form of ObtainInfoA

  SYNOPSIS
    error = ObtainInfoA(engineHandle, tagList)
                        A0            A1

    ULONG ObtainInfoA(struct GlyphEngine *, struct TagItem *);

    error = ObtainInfo(engineHandle, firstTag, ...)

    ULONG ObtainInfo(struct GlyphEngine *, Tag, ...);

  FUNCTION
    This function accepts a tagList whose tag field elements are
    valid for inquiry, and whose associated data fields are
    pointers to the destination in which to place the requested
    data.

    Tag items that refer to data indirectly (OT_Indirect is set)
    return pointers that may be allocated or cached by the
    library.  This data must be treated as read-only data.  When
    the application is done with the data acquired via ObtainInfoA,
    it must perform a ReleaseInfoA to allow the library to release
    the data.

  INPUTS
    engineHandle -- the handle acquired via OpenEngine.
    tagList -- a tagList containing OT_ tags valid for inquiry
          paired with the destination pointers for the inquiry
          results.  All destinations are longwords, whether they
          are pointers or values, and regardless of whether the
          value could fit in a smaller variable.

  RESULT
    This function returns a zero success indication, or a non-zero
    error code.

  EXAMPLE
        ULONG pointSize;
        struct Glyph *glyph;
        ...
        if (!ObtainInfo(EngineHandle, OT_Glyph, &glyph, TAG_DONE)) {
          ...
          ReleaseInfo(EngineHandle, OT_Glyph, glyph, TAG_DONE);
        }

  SEE ALSO
    ReleaseInfoA(), libraries/diskfonttag.h, libraries/oterrors.h
```

```
bullet.library/OpenEngine

  NAME
    OpenEngine -- Acquire engine handle

  SYNOPSIS
    engineHandle = OpenEngine()

    struct GlyphEngine *OpenEngine(void)

  FUNCTION
    This function establishes a context for access to the bullet
    library.  This context remains valid until it is closed via
    CloseEngine.  Each specific context isolates the specification
    of the various font attributes from other contexts concurrently
    accessing the bullet library.  A context can be shared among
    different tasks.

  RESULT
    This function returns an engineHandle, or NULL if for some
    reason no engineHandle can be created.

  EXAMPLE
        BulletBase = OpenLibrary("bullet.library", 0);
        if (!BulletBase)
          EndGame(ERROR_LibOpen, "bullet.library", 0);
        EngineHandle = OpenEngine();
        if (!EngineHandle)
          EndGame(ERROR_InternalCall, "OpenEngine");

  SEE ALSO
    CloseEngine()
```

```
bullet.library/ReleaseInfoA

    NAME
      ReleaseInfoA -- Release data obtained with ObtainInfoA
      ReleaseInfo -- varargs form of ReleaseInfoA

    SYNOPSIS
      error = ReleaseInfoA(engineHandle, tagList)
                           A0            A1

      ULONG ReleaseInfoA(struct GlyphEngine *, struct TagItem *);

      error = ReleaseInfo(engineHandle, firstTag, ...)

      ULONG ReleaseInfo(struct GlyphEngine *, Tag, ...);

    FUNCTION
      This function releases the data obtained with ObtainInfoA.
      Data associated with tags that are not indirect, i.e. for which
      OT_Indirect is not set, need not be released, but it is not an
      error to do so.  Released data may be immediately freed or may
      become a candidate to be expunged from memory when the system
      reaches a low memory condition, depending on the library's
      internal implementation.

      Each ReleaseInfoA tag item must be associated with a prior
      ObtainInfoA.  It is inappropriate to explicitly ReleaseInfoA a
      glyph that has only been implicitly obtained via GetGlyphMap --
      the glyph obtained via GetGlyphMap must only be released via a
      subsequent GetGlyphMap or via CloseEngine.

    INPUTS
      engineHandle -- the handle acquired via OpenEngine.
      tagList -- a tagList containing OT_ tags valid for inquiry
            paired with the data previously acquired for them with
            ObtainInfoA.  Null pointers quietly accepted and
            ignored for indirect data.

    RESULT
      This function has no result.  The only error that can occur is
      when the Obtain and Release pairs are mismatched: the
      application is assumed not to do that.

    EXAMPLE
          ULONG pointSize;
          struct Glyph *glyph;
          ...
          error = ObtainInfo(EngineHandle, OT_Glyph, &glyph, TAG_DONE);
          ...
          ReleaseInfo(EngineHandle, OT_Glyph, glyph, TAG_DONE);

    SEE ALSO
      ReleaseInfoA(), libraries/diskfonttag.h, libraries/oterrors.h
```

```
bullet.library/SetInfoA

    NAME
      SetInfoA -- Set font and/or glyph metrics
      SetInfo -- varargs form of SetInfoA

    SYNOPSIS
      error = SetInfoA(engineHandle, tagList)
                       A0            A1

      ULONG SetInfoA(struct GlyphEngine *, struct TagItem *);

      error = SetInfo(engineHandle, firstTag, ...)

      ULONG SetInfo(struct GlyphEngine *, Tag, ...);

    FUNCTION
      This function accepts a tagList whose tag field elements are
      valid for specification, and whose associated data fields are
      used to supply the specified data.

      Data that is supplied via an indirect pointer (OT_Indirect) to
      an array or structure is copied from that array or structure
      into the internal memory of the library.  Changes to the data
      after this call do not affect the engine.

    INPUTS
      engineHandle -- the handle acquired via OpenEngine.
      tagList -- a tagList containing OT_ tags valid for
            specification paired with the specification data.

    RESULT
      This function returns a zero success indication, or a non-zero
      error code.

    EXAMPLE
          if (!(error = SetInfo(EngineHandle, OT_PointHeight, fpoints,
                OT_GlyphCode, GC_daggerdbl, TAG_DONE)) {
            error = ObtainInfo(EngineHandle, OT_Glyph, &glyph);
            ...
            ReleaseInfo(EngineHandle, OT_Glyph, glyph);
          }

    SEE ALSO
      GetGlyphMap(), libraries/diskfonttag.h, libraries/oterrors.h,
```

Appendix B

```
#ifndef  LIBRARIES_DISKFONTTAG_H
#define  LIBRARIES_DISKFONTTAG_H
/*
**    $Id: diskfonttag.h,v 10.0 91/04/12 20:23:04 kodiak Exp $
**
**    libraries/diskfonttag.h -- tag definitions for .otag files
**
**    (C) Copyright 1990 Robert R. Burns
**        All Rights Reserved
*/

/* Level 0 entries never appear in the .otag tag list, but appear in font
 * specifications */
#define  OT_Level0     TAG_USER
/* Level 1 entries are required to exist in the .otag tag list */
#define  OT_Level1     (TAG_USER | 0x1000)
/* Level 2 entries are optional typeface metric tags */
#define  OT_Level2     (TAG_USER | 0x2000)
/* Level 3 entries are required for some OT_Engines */
#define  OT_Level3     (TAG_USER | 0x3000)
/* Indirect entries are at (tag address + data offset) */
#define  OT_Indirect   0x8000

/*******************************************************************/
/* font specification and inquiry tags */

/* !  tags flagged with an exclaimation mark are valid for
 *    specification.
 *  ? tags flagged with a question mark are valid for inquiry
 *
 * fixed binary numbers are encoded as 16 bits of integer and
 * 16 bits of fraction.  Negative values are indicated by twos
 * complement of all 32 bits.
 */

/* !  OT_DeviceDPI specifies the target device dots per inch -- X DPI is
 *    in the high word, Y DPI in the low word. */
#define  OT_DeviceDPI  (OT_Level0 | 0x01)/* == TA_DeviceDPI */

/* !  OT_DotSize specifies the target device dot size as a percent of
 *    it's resolution-implied size -- X percent in high word, Y percent
 *    in low word. */
#define  OT_DotSize      (OT_Level0 | 0x02)

/* !  OT_PointHeight specifies the requested point height of a typeface,
 *    specifically, the height and nominal width of the em-square.
 *    The point referred to here is 1/72".  It is encoded as a fixed
 *    binary number. */
#define  OT_PointHeight (OT_Level0 | 0x08)

/* !  OT_SetFactor specifies the requested set width of a typeface.
 *    It distorts the width of the em-square from that specified by
 *    OT_PointHeight.  To compensate for a device with different
 *    horizontal and vertical resolutions, OT_DeviceDPI should be used
 *    instead.  For a normal aspect ratio, set to 1.0 (encoded as
 *    0x00010000).  This is the default value. */
#define  OT_SetFactor  (OT_Level0 | 0x09)

/* !  OT_Shear... specifies the Sine and Cosine of the vertical stroke
 *    angle, as two fixed point binary fractions.  Both must be specified:
 *    first the Sine and then the Cosine.  Setting the sine component
```

```
 *    changes the Shear to an undefined value, setting the cosine
 *    component completes the Shear change to the new composite value.
 *    For no shear, set to 0.0, 1.0 (encoded as 0x00000000, 0x00010000).
 *    This is the default value. */
#define  OT_ShearSin    (OT_Level0 | 0x0a)
#define  OT_ShearCos    (OT_Level0 | 0x0b)

/* !  OT_Rotate... specifies the Sine and Cosine of the baselin rotation
 *    angle, as two fixed point binary fractions.  Both must be specified:
 *    first the Sine and then the Cosine.  Setting the sine component
 *    changes the Shear to an undefined value, setting the cosine
 *    component completes the Shear change to the new composite value.
 *    For no shear, set to 1.0, 0.0 (encoded as 0x00010000, 0x00000000).
 *    This is the default value. */
#define  OT_RotateSin  (OT_Level0 | 0x0c)
#define  OT_RotateCos  (OT_Level0 | 0x0d)

/* !  OT_Embolden... specifies values to algorithimically embolden -- or,
 *    when negative, lighten -- the glyph.  It is encoded as a fixed point
 *    binary fraction of the em-square.  The X and Y components can be
 *    changed indendently.  For normal characters, set to 0.0, 0.0
 *    (encoded as 0x00000000, 0x00000000).  This is the default value. */
#define  OT_EmboldenX  (OT_Level0 | 0x0e)
#define  OT_EmboldenY  (OT_Level0 | 0x0f)

/* !  OT_PointSize is an old method of specifying the point size,
 *    encoded as (points * 16). */
#define  OT_PointSize  (OT_Level0 | 0x10)

/* !  OT_GlyphCode specifies the glyph (character) code to use with
 *    subsequent operations.  For example, this is the code for an
 *    OT_Glyph inquiry */
#define  OT_GlyphCode   (OT_Level0 | 0x11)

/* !  OT_GlyphCode2 specifies the second glyph code.  For example,
 *    this is the right glyph of the two glyphs of an OT_KernPair
 *    inquiry */
#define  OT_GlyphCode2 (OT_Level0 | 0x12)

/* !  OT_GlyphWidth specifies a specific width for a glyph.
 *    It sets a specific escapement (advance) width for subsequent
 *    glyphs.  It is encoded as a fixed binary fraction of the em-square.
 *    To revert to using the font-defined escapement for each glyph, set
 *    to 0.0 (encoded as 0x00000000).  This is the default value. */
#define  OT_GlyphWidth (OT_Level0 | 0x13)

/* !  OT_OTagPath and
/* !  OT_OTagList specify the selected typeface.  Both must be specified:
 *    first the Path and then the List.  Setting the path name changes
 *    changes the typeface to an undefined value, providing the List
 *    completes the typeface selection to the new typeface.  OTagPath
 *    is the null terminated full file path of the .otag file associated
 *    with the typeface.  OTagList is a memory copy of the processed
 *    contents of that .otag file (i.e. with indirections resolved).
 *    There are no default values for the typeface. */
#define  OT_OTagPath    (OT_Level0 | OT_Indirect | 0x14)
#define  OT_OTagList    (OT_Level0 | OT_Indirect | 0x15)

/*  ? OT_GlyphMap supplies a read-only struct GlyphMap pointer that
 *    describes a bitmap for a glyph with the current attributes. */
#define  OT_GlyphMap    (OT_Level0 | OT_Indirect | 0x20)

/*  ? OT_WidthList supplies a read-only struct MinList of struct
 *    GlyphWidthEntry nodes for glyphs that are defined from GlyphCode
 *    to GlyphCode2, inclusive.  The widths are represented as fixed
 *    binary fractions of the em-square, ignoring any effect of
 *     SetFactor or GlyphWidth.  A width would need to be converted to
 *     a distance along the baseline in device units by the
 *     application. */
#define  OT_WidthList  (OT_Level0 | OT_Indirect | 0x21)

/*  ? OT_...KernPair supplies the kern adjustment to be added to the
 *     current position after placement of the GlyphCode glyph and
 *     before placement of the GlyphCode2 glyph.  Text kern pairs are
 *     for rendering body text.  Display kern pairs are generally
 *     tighter values for display (e.g. headline) purposes.  The
 *     adjustment is represented as a fixed binary fraction of the
 *     em-square, ignoring any effect of SetFactor.  This number would
 *     need to be converted to a distance along the baseline in device
 *     units by the application. */
#define  OT_TextKernPair (OT_Level0 | OT_Indirect | 0x22)
#define  OT_DesignKernPair (OT_Level0 | OT_Indirect | 0x23)



/*********************************************************************/
/* .otag tags */

/* suffix for files in FONTS: that contain these tags */
#define  OTSUFFIX".otag"

/* OT_FileIdent both identifies this file and verifies its size.
 * It is required to be the first tag in the file. */
#define  OT_FileIdent  (OT_Level1 | 0x01)

/* OT_Engine specifies the font engine this file is designed to use */
#define  OT_Engine     (OT_Level1 | OT_Indirect | 0x02)
#define  OTE_Bullet    "bullet"

/* OT_Family is the family name of this typeface */
#define  OT_Family     (OT_Level1 | OT_Indirect | 0x03)

/* The name of this typeface is implicit in the name of the .otag file */
/* OT_BName is used to find the bold variant of this typeface */
#define  OT_BName (OT_Level2 | OT_Indirect | 0x05)
/* OT_IName is used to find the italic variant of this typeface */
#define  OT_IName (OT_Level2 | OT_Indirect | 0x06)
/* OT_BIName is used to find the bold italic variant of this typeface */
#define  OT_BIName     (OT_Level2 | OT_Indirect | 0x07)

/* OT_SymSet is used to select the symbol set that has the OT_YSizeFactor
 * described here.  Other symbol sets might have different extremes */
#define  OT_SymbolSet  (OT_Level1 | 0x10)

/* OT_YSizeFactor is a ratio to assist in calculating the Point height
 * to BlackHeight relationship -- high word: Point height term, low
 * word: Black height term -- pointSize = ysize*<high>/<low> */
#define  OT_YSizeFactor (OT_Level1 | 0x11)

/* OT_SpaceWidth specifies the width of the space character relative
 * to the character height */
#define  OT_SpaceWidth (OT_Level2 | 0x12)

/* OT_IsFixed is a boolean indicating that all the characters in the
 * typeface are intended to have the same character advance */
#define  OT_IsFixed    (OT_Level2 | 0x13)

/* OT_SerifFlag is a boolean indicating if the character has serifs */
#define  OT_SerifFlag  (OT_Level1 | 0x14)

/* OT_StemWeight is an unsigned byte indicating the weight of the character
*/
#define  OT_StemWeight (OT_Level1 | 0x15)

#define  OTS_UltraThin   8/*   0- 15 */
#define  OTS_ExtraThin  24/*  16- 31 */
#define  OTS_Thin 40    /*  32- 47 */
#define  OTS_ExtraLight 56/*  48- 63 */
#define  OTS_Light      72/*  64- 79 */
#define  OTS_DemiLight  88/*  80- 95 */
#define  OTS_SemiLight 104/*  96-111 */
#define  OTS_Book120    /* 112-127 */
#define  OTS_Medium    136/* 128-143 */
#define  OTS_SemiBold  152/* 144-159 */
#define  OTS_DemiBold  168/* 160-175 */
#define  OTS_Bold184    /* 176-191 */
#define  OTS_ExtraBold 200/* 192-207 */
#define  OTS_Black     216/* 208-223 */
#define  OTS_ExtraBlack232/* 224-239 */
#define  OTS_UltraBlack248/* 240-255 */

/* OT_SlantStyle is an unsigned byte indicating the font posture */
#define  OT_SlantStyle (OT_Level1 | 0x16)
#define  OTS_Upright   0
#define  OTS_Italic    1/* Oblique, Slanted, etc. */
#define  OTS_LeftItalic2/* Reverse Slant */

/* OT_HorizStyle is an unsigned byte indicating the appearance width */
#define  OT_HorizStyle (OT_Level1 | 0x17)
#define  OTH_UltraCompressed 16/*   0- 31 */
#define  OTH_ExtraCompressed 48/*  32- 63 */
#define  OTH_Compressed 80/*  64- 95 */
#define  OTH_Condensed 112/*  96-127 */
#define  OTH_Normal    144/* 128-159 */
#define  OTH_SemiExpanded176/* 160-191 */
#define  OTH_Expanded  208/* 192-223 */
#define  OTH_ExtraExpanded240/* 224-255 */

/* OT_SpaceFactor specifies the width of the space character relative
 * to the character height */
#define  OT_SpaceFactor(OT_Level2 | 0x18)

/* OT_InhibitAlgoStyle indicates which ta_Style bits, if any, should
 * be ignored even if the font does not already have that quality.
 * For example, if FSF_BOLD is set and the typeface is not bold but
 * the user specifies bold, the application or diskfont library is
 * not to use OT_Embolden to achieve a bold result. */
#define  OT_InhibitAlgoStyle (OT_Level2 | 0x19)

/* OT_AvailSizes is an indirect pointer to sorted UWORDs, 0th is count */
#define  OT_AvailSizes (OT_Level1 | OT_Indirect | 0x20)
#define  OT_MAXAVAILSIZES20/* no more than 20 sizes allowed */

/* OT_SpecCount is the count number of parameters specified here */
#define  OT_SpecCount  (OT_Level1 | 0x100)

/* Specs can be created as appropriate for the engine by ORing in the
 * parameter number (1 is first, 2 is second, ... up to 15th) */
#define  OT_Spec  (OT_Level1 | 0x100)
/* OT_Spec1 is the (first) parameter to the font engine to select
 * this particular typeface */
#define  OT_Spec1 (OT_Level1 | 0x101)

#endif   /* LIBRARIES_DISKFONTTAG_H */
```

```
#ifndef  LIBRARIES_GLYPH_H
#define  LIBRARIES_GLYPH_H
/*
**     $Id: glyph.h,v 9.0 91/04/09 20:02:37 kodiak Exp $
**
**     libraries/glyph.h -- structures for glyph libraries
**
**     (C) Copyright 1991 Robert R. Burns
**         All Rights Reserved
*/

#ifndef  EXEC_NODES_H
#include <exec/nodes.h>
#endif

/* A GlyphEngine must be acquired via OpenEngine and is read-only */
struct GlyphEngine {
    struct Library *gle_Library; /* engine library */
    char *gle_Name;     /* library basename: e.g. "bullet" */
    /* private library data follows... */
};

typedef LONG FIXED;     /* 32 bit signed w/ 16 bits of fraction */

struct GlyphMap {
    UWORD   glm_BMModulo;/* # of bytes in row: always multiple of 4 */
    UWORD   glm_BMRows;/* # of rows in bitmap */
    UWORD   glm_BlackLeft;/* # of blank pixel columns at left */
    UWORD   glm_BlackTop;/* # of blank rows at top */
    UWORD   glm_BlackWidth;/* span of contiguous non-blank columns */
    UWORD   glm_BlackHeight;/* span of contiguous non-blank rows */
    FIXED   glm_XOrigin;/* distance from upper left corner of bitmap */
    FIXED   glm_YOrigin;/*   to initial CP, in fractional pixels */
    WORD    glm_X0;     /* approximation of XOrigin in whole pixels */
    WORD    glm_Y0;     /* approximation of YOrigin in whole pixels */
    WORD    glm_X1;     /* approximation of XOrigin + Width */
    WORD    glm_Y1;     /* approximation of YOrigin + Width */
    FIXED   glm_Width; /* character advance, as fraction of em width */
    UBYTE  *glm_BitMap;/* actual glyph bitmap */
};

struct GlyphWidthEntry {
    struct MinNode gwe_Node;/* on list returned by OT_WidthList inquiry */
    UWORD   gwe_Code;   /* entry's character code value */
    FIXED   gwe_Width; /* character advance, as fraction of em width */
};
#endif   /* LIBRARIES_GLYPH_H */
```

Debugging Tools - Choosing the Right Tool for the Job

by Carolyn Scheppner

Note: Some of the debugging tools supplied on the Devcon disk are from the CATS support item "Software Toolkit". Full instructions on the use and options of some of the more complex tools such as *Wack* and *Wedge* may be found in the Software Toolkit manual.

General Warning: Some debugging tools stress the system, or allow you to wedge into arbitrary system routines, or attempt to provoke improperly written code to crash. We have attempted to mark these kinds of tools with warnings below. You probably should not write to un-backed-up disks or harddisks while using such tools.

Unless otherwise noted, the following tools are all Copyright (c) 1985, 1990, 1991 Commodore-Amiga. Inc. All Rights Reserved. They are provided for debugging purposes only and may not be redistributed in any manner.

Watchdog Tools

Watchdog tools help trap illegal memory accesses. Such accesses are generally caused by using improperly initialized variables or structures, or by accessing structures and memory that have already been freed. Code with illegal accesses may appear to run fine under most circumstances but may fail or crash unexpectedly in the field.

Unfortunately, it is currently not possible to trap all illegal accesses. If a program is accessing or trashing memory in normal legal user memory spaces, or even trashing itself, these tools won't catch it in the act. Luckily, a majority of illegal accesses reference low memory or freed memory. By using a freed memory invalidation tool like *MungWall* in conjunction with an illegal access watchdog tool, the majority of these problems can be caught.

The best watchdog tools require an MMU. Processor-based tools such as *MemWatch* and *WatchMem* can watch for writes to low memory. But they can't catch reads of low memory or other illegal accesses.

MMU-based watchdogs such as *Enforcer* and *CPU* can trap all illegal accesses of low memory, non-existent memory, and ROM, reporting the exact type of access, as well as the offending code's program counter and registers. The debugging information is sent to a serial terminal (or parallel printer with *CPU's cputrap.par*).

If the illegal access occurs in ROM code, you can generally trace forward on the stack to find the program address that called the ROM routine. It is then possible to disassemble a program in memory at the point it caused the illegal access. Programmers who like to debug at a low level may then either immediately recognize the problem, or can compare the code disassembled in memory to disassemblies of their object modules (or to their source code if the source is in assembler).

Programmers who prefer to debug at a higher level can compile a debugging version of their software to allow them to track which code is executing when the illegal access occurs. This can be accomplished by stepping or breakpointing with a debugger, or by inserting remote debugging statements (kprintf() or dprintf()) to the same remote device that is receiving the watchdog output. Plain printf() debugging could also be used with Delay()'s to allow time for watching both the printf() debugging and the remote watchdog output.

All software should be tested with a memory invalidator, such as *MungWall*, running in conjunction with one of the illegal access trappers. It is extremely useful to use such tools while you are developing so that you can catch illegal accesses right away - they are much easier to find without disassembly if you just wrote or changed the code.

```
TOOLNAME: Enforcer and Enforcer.par
CATEGORY: MMU-based Watchdog tool
USAGE:    Enforcer [off|on|quiet|fprotect]
USED FOR: Trapping reads and writes of low/non-existent memory
REQUIRES: MMU that is not being used, serial terminal or parallel printer
FOUND ON: Devcon disk
```

```
TOOLNAME: Lawbreaker
CATEGORY: Test program for Enforcer
USAGE:    Lawbreaker
REQUIRES: CPU or Enforcer
FOUND ON: Devcon disk
```

```
TOOLNAME: WatchMem  (inspired by MemWatch by John Toebes VIII)
CATEGORY: Processor-based low memory watchdog
USAGE:    RUN Watchmem [ file | window ] opt n [interval ]
          (opt n = nocorrect)
USED FOR: Trapping writes to low memory
WARNINGS: This processor-based tool can not prevent writes to low memory.
          It can correct them after they occur, but you might crash first.
          If you have an MMU, use Enforcer instead!
FOUND ON: Devcon disk, Software Toolkit
```

TOOLNAME: MungWall (combination MemMung + MemWall + Snoop ++)
CATEGORY: Memory invalidation and memory overwrite/underwrite monitor
USAGE: Mungwall [UPDATE] [TASK name][WAIT] [NOWAIT][SNOOP] [NOSNOOP] [INFO][PRESIZE] [POSTSIZE [FILLCHAR]
USED FOR: Catching accesses of uninitialized and freed memory. Finding things that write past their allocated memory. Also for general snooping of memory allocations by one or all tasks. (See also SnoopStrip for filtering of snoop option output)
REQUIRES: Serial terminal
WARNINGS: Will provoke bad code to crash if not used with Enforcer/CPU
FOUND ON: Devcon disk

TOOLNAME: MemMung
CATEGORY: Memory invalidation tool (more pleasant with Enforcer/CPU)
USAGE: RUN MemMung
USED FOR: Catching accesses of uninitialized and freed memory
WARNINGS: Will provoke bad code to crash if not used with Enforcer/CPU
FOUND ON: Devcon disk, Software Toolkit

TOOLNAME: MemWall
CATEGORY: Memory allocation overwrite/underwrite monitor
USAGE: Memwall [all] [fill N] [presize N] [postsize N] [snoop] [supersnoop]
USED FOR: Finding things that write past their allocated memory. Also for general snooping of memory allocations.
REQUIRES: Serial terminal
WARNINGS: Some things in the system (such as layers) free memory in smaller chunks than they allocated. When this is done, (or when it finds a fill area hit), it does NOT let that area actually be deallocated. This can lead to loss of memory. Note that presize or postsize may be 0.
FOUND ON: Devcon Disk

TOOLNAME: Scratch (by Bill Hawes)
CATEGORY: Scratch register (D1,A0,A1) invalidation tool
USAGE: See sample script scratch_all
USED FOR: Will provoke code with improper register usage to fail
FOUND ON: Devcon Disk (if available in time)

TOOLNAME: IO_Torture
CATEGORY: Specialized watchdog for IORequest re-use
USAGE: IO_Torture
USED FOR: Remote monitoring of premature re-use of IORequests
REQUIRES: Serial terminal
FOUND ON: Devcon disk

Stressing Tools

Some tools are designed to cause stressful conditions for your software such as low memory and emulation of a heavy multitasking environment. Testing your software while running such tools can help turn up faulty or missing error checking code, and race conditions.

TOOLNAME: EatMem
CATEGORY: Low memory test tool
USAGE: EatMem (adjust gadgets for desired amount/sizes of free memory)
USED FOR: Testing software behavior under low memory conditions
FOUND ON: Devcon disk

TOOLNAME: EatCycles
CATEGORY: Multitasking load test tool
USAGE: EatCycles (adjust gadgets for desired load)
USED FOR: Testing software behavior under heavy system load
FOUND ON: Devcon disk

TOOLNAME: Memoration (by Bill Hawes)
CATEGORY: Low memory test tool
USAGE: See doc file
USED FOR: Selectively restricting available memory
FOUND ON: Devcon Disk (if available in time)

Monitoring Tools

TOOLNAME: Tstat
CATEGORY: Task monitor
USAGE: Tstat [CLI# | ExecTaskName | CliCommandName] [-tickdelay]
USED FOR: Monitoring PC, regs, stack, signals, etc. of a running task
FOUND ON: Devcon disk

TOOLNAME: Wedge
CATEGORY: System function monitor
USAGE: Complex and best done with scripts - type Wedge help for help
USED FOR: Monitoring the calls to and results from any system function
REQUIRES: Limited local monitoring, serial or parallel for full monitoring
WARNINGS: Can bog system down; can crash if calling task has tiny stack. Local monitoring can cause recursive looping if functions called by text output routines are wedged.
FOUND ON: Devcon disk, Software Toolkit (full instructions with ToolKit)

TOOLNAME: DevMon (see DEVICE TOOLS)

Crash Trapping Tools

TOOLNAME: SRT
CATEGORY: Software error trapping wedge (for 1.2/1.3, not 2.0)
USAGE: SRT [srt.textfile] (default s:srt.text)
USED FOR: Examining name, registers, PC, SP, of crashed task
FOUND ON: Devcon Disk

TOOLNAME: TNT
CATEGORY: Software error trap handler (for all version of OS)
USAGE: TNT (must be installed before the crash occurs)
USED FOR: Examining name, registers, PC, SP, of crashed task
WARNINGS: You may need to do TNT OFF before using a trap-based debugger.
FOUND ON: Devcon disk

General Debuggers and Disassemblers

Many development language packages come with excellent source level debuggers and object module disassemblers. In addition, the following tools are useful for debugging executables:

TOOLNAME: Wack (Originated by Carl Sassenrath)
CATEGORY: Symbolic debugger/disassembler
USAGE: Wack "program [programargs]" (see SW Toolkit for other opts)
USED FOR: Disassembling, single stepping, breakpointing
WARNINGS: Improper use could lead to a crash. Wack1.0 installs/leaves a trap handler. If used with TNT, RUN Wack so only the handler of the bg run process will be changed.
FOUND ON: Devcon disk, Software Toolkit

TOOLNAME: RomWack
CATEGORY: ROM-based debugger
USAGE: Enter with exec Debug() function or RomWack command (SW Toolkit)
USED FOR: Freezing the Amiga while you examine memory remotely
REQUIRES: Serial terminal
WARNINGS: Improper use could lead to a crash.
FOUND IN: the Amiga OS

TOOLNAME: Metascope (by Metadigm)
CATEGORY: Multiwindow Intuition interface symbolic debugger/disassembler
USED FOR: Disassembling, single stepping, breakpointing
WARNINGS: Improper use could lead to a crash.
FOUND IN: Stores (Commercial product)

System Configuration Listers

Configuration listers are handy for checking the address, version, or presence of various system hardware and software items. If you are working with devices or libraries, you can use the memory tool *Flush* to flush your device or library from memory and *LibList* or *DevList* will check that the device or library has actually been removed from the system. *Config* can be used to check a machine's OS and custom chip versions, processor type, and configured devices without taking off the cover.

```
TOOLNAME: Config
CATEGORY: Motherboard and Autoconfig configuration lister
USAGE:    Config [debug]
USED FOR: Checking ROM/Processor/Chip versions, and autoconfig devices
          With debug option, for checking all autoconfig params of boards.
FOUND ON: Devcon disk


TOOLNAME: TaskList, LibList, DevList, ModList, and (C. Sassenrath) IntList
CATEGORY: System software list display tools
USAGE:    No arguments for any of these
USED FOR: Checking address, version, presence, of tasks, libs, devs, etc.
FOUND ON: Devcon disk (see also Memory Tool "Flush")


TOOLNAME: MemList - see MEMORY TOOLS


TOOLNAME: DosList - see DOS/DISK TOOLS
```

Memory Tools

Most memory tools are used to check for, and debug memory losses and other memory allocation and deallocation problems. *Avail* and *Flush* can be used together to make sure that an application is freeing all of its memory. *Flush* is required because libraries, devices, and fonts loaded from disk will hang around in memory even after they have been closed until someone asks for the memory.

To check your application for memory loss, arrange your Workbench so that you have an open shell (for *Avail*) and can start your application from a different shell or from an icon without rearranging any windows (rearranging windows causes memory fluctuations). If possible, size the shell window for *Avail* tall enough for the output of two *Avails* and a couple of *Flushes* (so that you won't have to write down any numbers).

Then, without rearranging any windows, do:

1. *Flush*
2. *Avail* (note these pre-application Available totals)
3. Start your application
4. [optional *Avail* here to check run-time memory usage]
5. Exit your application
6. *Flush*
7. *Avail* (the Available totals should match the pre-application ones)

TOOLNAME: Avail
CATEGORY: Memory free/largest lister
USAGE: Avail (2.0 has flush opt; use Flush command with earlier Avails)
USED FOR: Checking memory usage, and memory loss in conjunction with Flush
FOUND ON: Workbench

TOOLNAME: Flush
CATEGORY: Memory flusher (to check for real memory loss)
USAGE: Flush (Note - Flush does 3 flushes when invoked)
USED FOR: Flushing all currently unused devices/libraries/fonts from memory
FOUND ON: Devcon disk (use in conjunction with Avail)

TOOLNAME: MemMon
CATEGORY: Memory use recorder (helps narrow search for lost memory)
USAGE: MemMon (>diskfile)
USED FOR: Producing a commented record of memory usage
FOUND ON: Devcon disk

TOOLNAME: Frags
CATEGORY: Memory fragmentation summarizer
USAGE: Frags [full]
USED FOR: Checking for memory fragmentation.
FOUND ON: Devcon disk, Software Toolkit

TOOLNAME: MemList
CATEGORY: Full used and free memory chunk lister
USAGE: Memlist [>diskfile]
USED FOR: Debugging fragmentation/deallocation problems
FOUND ON: Devcon disk, Extras(?)

TOOLNAME: Owner
CATEGORY: Memory ownership tool
USAGE: Owner [0x] nnnn... (owner ? for help)
USED FOR: Trying to determine ownership of allocated memory
FOUND ON: DevCon disk

TOOLNAME: Snoop
CATEGORY: Remote AllocMem/FreeMem debugger
USAGE: Snoop (use SnoopStrip on captured output to isolate unfreed Allocs)
USED FOR: Debugging unfreed memory problems
REQUIRES: Serial terminal
FOUND ON: Devcon disk, Software Toolkit

TOOLNAME: SnoopStrip
USAGE: SnoopStrip [>outfile] infile
USED FOR: Stripping matched allocs/frees from captured snoop output
FOUND ON: Devcon disk

TOOLNAME: Drip
CATEGORY: Memory loss accumulator
USAGE: Drip [threshold]
USED FOR: Determining change in free memory since last invocation
FOUND ON: Devcon disk, Software Toolkit

TOOLNAME: Peek
USAGE: Peek B|W|L [0x]address [[0x]compvalue] [[0x]mask]
USED FOR: Checking or script branching on contents of a memory address
FOUND ON: Devcon disk

TOOLNAME: Poke
USAGE: Peek B|W|L [0x]address [0x]value [[0x]mask]
USED FOR: Changing the contents of a memory address
WARNINGS: Obviously, poking where you shouldn't may crash machine.
FOUND ON: Devcon disk

OS/Disk Checking Tools

TOOLNAME: DosList
CATEGORY: Dos device lister
USAGE: Doslist [DEVS|VOLS|DIRS]
USED FOR: Examining the dos device list
FOUND ON: Devcon disk

TOOLNAME: ShowLocks (Copyright 1988 Chuck McManis)
CATEGORY: Filelock lister
USAGE: ShowLocks [volumename:]
USED FOR: Displaying outstanding locks
FOUND ON: Software Toolkit

TOOLNAME: DiskEd
CATEGORY: Disk sector editor
USAGE: DiskEd drivename: (see Bantam AmigaDos manual, 2nd Edition, for instructions)
USED FOR: Examining and modifying disk sector data
WARNINGS: Improper use can trash disk data or structure.
FOUND ON: Devcon Disk

Performance Checking Tools

TOOLNAME: PerfMon
CATEGORY: System performance monitor
USAGE: Perfmon
USED FOR: Checking for busy waiting and other performance problems
FOUND ON: Devcon Disk, 1.3 Extras

Intuition/Graphics Tools

TOOLNAME: WinList
USED FOR: Examining addresses, titles, flags, sizes of screens and windows
FOUND ON: Devcon disk

TOOLNAME: ShowGfxBase
USED FOR: Examining GfxBase normal display sizes and flags
FOUND ON: Devcon disk

TOOLNAME: ReadPixel
USED FOR: Reading the XY screen location and color of pixels. Can be used to check the size and position of onscreen images.
USAGE: Readpixel (then click on pixels to read)
FOUND ON: Devcon disk

TOOLNAME: MKS_Lens
USED FOR: Examining individual pixels of screen
USAGE: MKS_lens
FOUND ON: Devcon disk

TOOLNAME: DevMon
CATEGORY: Device monitor
USAGE: Devmon name.device unitnum [remote] [hex] [allunits] [full]
(remote is serial output, full has exec wedges in DoIO, ReplyMsg)
USED FOR: Monitoring the calls to a device
REQUIRES: Nothing for local monitoring. Serial terminal optional (slower)
WARNINGS: Stresses system if wedged into high-usage or time-critical devices (such stress could lead to crashes or hangs)
FOUND ON: Devcon disk

TOOLNAME: Cmd
CATEGORY: Parallel/Serial output capture tool
USAGE: Type cmd help for usage
USED FOR: Debugging printer, serial, and parallel output
FOUND ON: Workbench 2.0

TOOLNAME: PrinterTest
CATEGORY: Printer driver test suite
USAGE: Printertest (then answer y when the correct printer is prompted)
USED FOR: Testing printer drivers
FOUND ON: Devcon disk

TOOLNAME: KTest
CATEGORY: Serial connection tester
USAGE: KTest
USED FOR: Checking serial debugging setup
FOUND ON: Devcon disk

TOOLNAME: DTest
CATEGORY: Parallel connection tester
USAGE: DTest
USED FOR: Checking parallel debugging setup
FOUND ON: Devcon disk

TOOLNAME: IO_Torture (see WATCHDOG TOOLS)

Development Time Bug Prevention/Tracking Tools

TOOLNAME: RCS
CATEGORY: Source/document control
USAGE: See accompanying docs on Fish Disk 282
USED FOR: Recording changes to source code and documents
FOUND ON: Fish Disk 282

TOOLNAME: Autodoc
CATEGORY: Source code autodoc extractor
USAGE: See accompanying autodoc.doc
USED FOR: Extracting standard function documentation from your source code
FOUND ON: Devcon disk

TOOLNAME: Bumprev
CATEGORY: Revision bumper
USAGE: Bumprev version revname (example: bumprev 36 my_rev)
USED FOR: Updating revision include files (xxx_rev.h and xxx_rev.i)
FOUND ON: Devcon disk

◆

CDTV Tools

by Ben Phister

Portions of this article are taken from "The CTrac CD-ROM Emulation for the Commodore CDTV" user manual, courtesy of ICOM Simulations; Xiphias database documentation courtesy of Xiphias, Inc.; and Timefold compression documentation courtesy of Silent Software.

Overview

One of the major advantages in developing multimedia applications for CDTV is the rich development environment, inherited from over 5 years of Amiga developer activities. Compilers, editors, paint packages, video tools, authoring systems, audio and video capture systems -- all are available in abundance.

However, developing for CDTV requires further tools to let the developer benefit fully from the possibilities (and overcome the limitations) of the new technologies. This paper will discuss some of the more important tools which are currentlly available or under development for CDTV:

☐ The CTrac Emulation system allows for complete emulation of a CDTV application, using the SCSI hard disk in an Amiga instead of a CD-ROM.

☐ The Xearch database and search engine, from Xiphias, provides means to index entire textual databases, and create and manipulate hot-word links throughout the database.

☐ Time Fold, from Silent Software, is a video and animation compression system, allowing compression of standard ANIM files by up to a factor of 6.

☐ The CDXL Toolkit, from Sassenrath Research, provides a set of tools to create "video streams" which can be played back using Commodore's proprietary CD-XL technology, allowing CDTV to display 1/4 screen, 12 frames-per-second video in 6 bitplanes.

☐ The CDTV Tools disk contains numerous utilities and code samples showing important and useful techniques to debug applications, access CDTV specific features, etc.

☐ A number of authoring systems and languages have been or are being adapted for use with CDTV. These include The Director II, AmigaVision, and the AMOS language.

CTrac Emulator

Introduction

The CTrac CD-ROM emulation system, developed by ICOM Simulations, Inc., is a combination of hardware and software that emulates CD-ROM discs for the CDTV. The emulation system creates an image of a CD-ROM disc in a file on the hard disk drive of an Amiga. The system then acts as the CD-ROM drive by presenting the image to the CDTV, as if it were an actual CD-ROM disc. Since the emulation system replaces the CD-ROM drive in the CDTV, all formats and modes of CD-ROM can be emulated including CD-DA, CD+G and CD+MIDI.

This is a true emulation environment, not a simulation. The application actually runs in the memory of the CDTV. The emulator board replaces the CD-ROM mechanism. Every time the CDTV tries to access the CD-ROM, it accesses the emulator board. The board and the emulation software in turn access an ISO 9660 disc image on the Amiga's SCSI disk drive. The board slows down the seek times of the SCSI drive, as well as the data transfer rate, to accurately emulate the performance of a CD-ROM drive.

Until now, the only accurate way to test CD-ROM applications was to have the application pressed on to a CD-ROM disc and then tested in the CDTV. The process of pressing a disc to test an aplication under development is time consuming and costly, especially if the application does not work properly. Emulating CD-ROM applications with the CTrac emulation system not only eliminates the need to have discs pressed for testing but also allows accurate monitoring of the commands sent to the CD-ROM drive. We do recommend that a test disc be cut before mastering the application, however.

The CTrac CD-ROM emulation system hardware consists of a printed circuit board that plugs into an Amiga 2500 or Amiga 3000, and an interface cable. The cable connects the emulator board to the CD-ROM drive connector in the CDTV. The CTrac hardware together with the CTrac software running on the Amiga emulates the CD-ROM drive in the CDTV. The emulation is completely transparent to the CDTV.

CTrac Contents and Software Description

Hardware. The CTrac emulation hardware consists of the CTrac emulation printed circuit board, and an interface cable.

Software. The CTrac emulation software consists of the following files and directories:

☐ Emulate	The emulation software
☐ Builddisc	A tool used to create entire CD disc images for the CTrac emulation system. It combines track images, along with the proper subcode information, into one composite disc image. The final disc image is needed for the emulation software.
☐ Buildtrack	A tool used to create ISO 9660 formatted tracks that are needed on the CD-ROM discs used in the CDTV. The output of this program is used an input for the BuildDisc program.

☐ ISOUtil	A tool used to extract, update and list directories of files contained in ISO track images or from disc images that contain ISO tracks.
☐ *Libs:*	The *libs:* directory contains the requestor library that must be installed on the Amiga before running the emulation software.
☐ QFS	The QFS directory contains further directories and files needed to run the Qwik File System on the Amiga. The Qwik File System was developed by CONSULTRON. The data rate and seek time requirements of the emulator exceed the abilities of the Amiga Fast File System. Therefore, the Qwik File System is provided and must be installed on the hard drive that will contain the disc images that will be used for emulation. The Qwik File System is approximately 16 times faster than the Fast File System when performing directory operations and approximately 2 times faster when reading files greater than 36K. The Qwik File System must be installed on the hard drive that will contain the disc images to be emulated. The Qwik File System is compatible with the AmigaDOS file requirements with the following exception: 16 character file/directory comments must be used instead of 79.
☐ Install	A batch file that copies the emulation software and tools onto the command directory of the Amiga boot drive.

Hardware Requirements

☐ Amiga 2500, Amiga 3000, or Amiga 3500, with one free expansion slot.

The CTrac emulation software was designed to run on an Amiga 2500 or higher model, such as the Amiga 3000. If an Amiga 2500 will be used for development, the emulation software must be run in the 2500's 68030 mode. 3 Mbytes of RAM minimum are required.

☐ 2 or 3 hard disk drives, with one 600 Mbytes or larger.

At least two hard disk drives should be used with the CTrac emulation software. One hard drive should utilize the Amiga Fast File System and contain all of the standard Amiga libraries and tools. The second hard drive, which should be the larger of the two, must have the Qwik File System installed. It will contain the disc images that will be emulated. If the image is too large, you may need a third drive.

CD-ROM discs may contain varying amounts of data up to a maximum of approximately 650 MB. Since the image of a CD-ROM disc is stored in a file on the hard drive of an Amiga, it follows that the hard drive used must be at least as large as the largest image that will be emulated. Moreover, it may be desireable to store the source data and track images that were used to build the disc image on the same hard drive. If this is done, it could more than double the amount of storage space required. Therefore, a 600 MB or larger drive is recommended.

For example, assume your application requires 400 Mbytes of data and code. In this case you would probably need 3 hard disk drives, or a total capacity of 1200 MBytes.

Drive 1, formatted under the Fast File System, contains your original source data and code. You develop and test your application on this drive.

Drive 2 will receive the track image built by the BuildTrack utility. This image will be at least as large as your application code + data (400 MB, in this case).

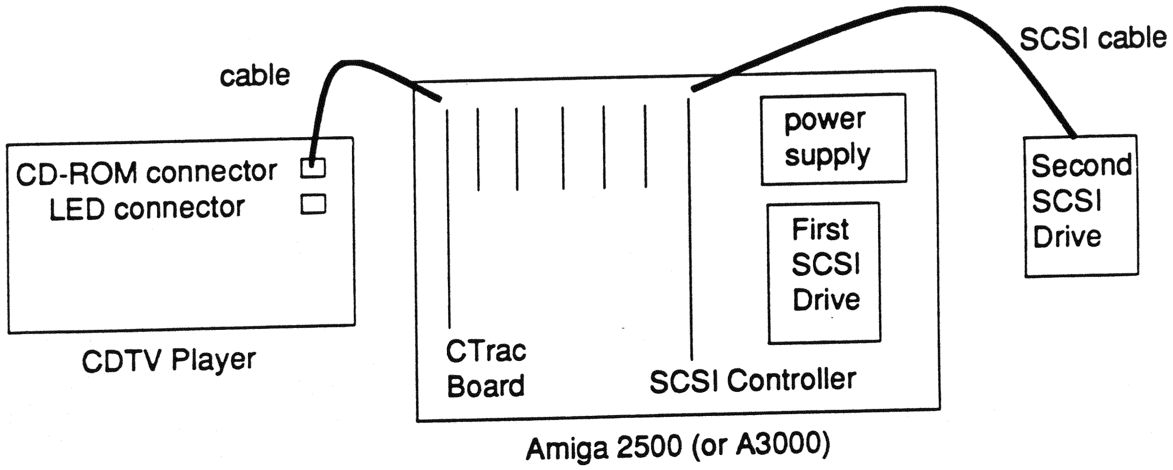
Drive 3, which must be formatted under the Qwik File System, will receive the disc image built by the BuildDisc utility. Again, this image will be at least as large as your application code (400 MB).

Note that you may combine both the track image and the disc image on one drive, if they will fit. If space allows, you can even build those images on your source drive, if you have formatted your source drive with the Qwik File System. Thus if your application contains only 30 Mbytes of code and data, you could develop it and test it under the emulator with a single 100 Mbyte drive.

The hard drive containing the disc image must also be reasonably fast because the emulator requires at least a 175K bytes/second maintainable data rate. For example, we have tested the drive using the Quantum 40 and 100 Mbyte drives delivered with Amiga 3000's. Unfortunately, these drives are not fast enough to allow proper emulation. The emulator reports error messages indicating that it could not read data fast enough from those drives. We have successfully used Seagate 450 Mbyte drives (reference ST2502N 94241-502) to resolve these problems.

Installation of the Emulation System

Installation of the system is fairly quick, requiring approximately 20 minutes. First, the large SCSI disk must be formatted under the Qwik File System. Next, the install program is executed to copy the tools necessary for image building and emulation to the Amiga hard disk drive. Finally, the CTrac emulation board must be installed in the Amiga 2500 or 3000, and connected to the CDTV player. The figure below indicates the physical set-up of the emulation system in an Amiga 2500.



Creating Disc Images and Using the Emulation Software

The following section describes the steps involved in preparing a disc image for the emulator, and using the emulator board and software to emulate that image.

For this discussion, we will assume that the application is in "mixed mode": that is, it contains a certain number of CD-DA audio tracks, as well as Amiga code and data. We will also assume the application has been written, and the data is ready. The application resides on a drive we shall name DH0: A second hard disk, named Q0:, has been formatted under the Qwik File System to receive the image files.

Step 1 ***Use the iso utility to create a controlfile for BuildTrack.*** The iso utility is provided on the ISO DevPak diskette, available to licenced CDTV developers. It reads the directory structure of your AmigaDOS drive (DH0:) and creates a file which specifies which files to include in the ISO 9660 track, and the order of those files. The iso utility allows files to be sorted by size, alphabetically by name, or to remain in the same order as they were on the source disk. Iso creates an ASCII text file, which the developer can modify for further optimizations.

Step 2 ***Use BuildTrack to create a track image.*** The BuildTrack utility reads in the controlfile created by the iso utility. It verifies for ISO 9660 compatibility (maximum levels of nesting for subdirectories, filename conventions, etc.) It then generates an ISO 9660 track image on the target drive (Q0:)

Step 3 ***Use the FixTM utility.*** The FixTM utility, also on the ISO DevPak diskette, modifies the ISO image to allow for direct booting on CDTV.

Step 4 ***Create the BuildDisc controlfile.*** Like the BuildTrack utility, BuildDisc is also directed by a control file. The control file, a standard text file, specifies which tracks will be included in the disc image, and in what order. It also allows other attributes of the disc to be specified, such as the length of the table of contents (TOC).

Step 5 ***Use BuildDisc to create the disc image.*** BuildDisc reads the controlfile, and generates a disc image with the track images that were specified.

Step 6 ***Run the emulator using the disc image.*** Now we can begin the actual emulation. The emulator software runs in the memory of the Amiga 2500 or 3000. It requests the name of the disc image file which should be emulated, and then begins to work.

Step 7 ***Reboot the CDTV.*** Finally, we power on (or reset) the CDTV. The emulator software (along with the emulator board) will intercept all attempts by the CDTV to access the CD-ROM drive. They access the disc image on the SCSI drive instead. Furthermore, the board and software assure emulation of the data transfer rate and seek times of the CD-ROM mechanism.

The CDTV application can be viewed on a monitor attached to one of the CDTV video outputs. Meanwhile, the emulator software displays activity messages in its Activity Window. All accesses to the ROM are indicated, as well as all commands received by the emulator.

A status window at the bottom of the screen displays the name of the image file being emulated, any emulator errors that have occured during emulation, and the current state of the drive status bits (ready, audio, done, error, SpinUp, DiscIn, InfErr).

Optionally, time emulation can be suspended. If this is done, the emulator no longer emulates spin up, spin down, and seek delays. This permits testing applications at the fastest possible speed, but the emulation is no longer accurate.

Emulator Limitations

☐ Seek Times	The emulator approximates the seek times and other delays of the CD-ROM drive in the CDTV. Due to the variances in drives and discs, it is possible that a seek time on the emulator could be slightly different from that on the actual drive.
☐ Tasks	In order to obtain optimal results, avoid running other tasks on the Amiga 2500 or 3000 simultaneously with the emulator software. The emulator needs a large amount of CPU time, and its priority is set to 100. Furthermore, contention may develop between two tasks trying to use the SCSI bus simultaneously.

Xearch Database and Search Engine

Overview

Xiphias has developed a database and search engine system, XDB-SE, to support its CD-ROM products. Xiphias uses the system for Apple Macintosh systems and for product delivery on Apple, MS-DOS and Commodore CDTV products.

A key concept in all the XDB_SE products is the use of inverted indices. These provide a simple but powerful hypertext linkage by searching the indices. The application defines the fields of that are to be inverted and the XDB-SE system handles the maintenance of the data.

Searching of the inversion indices is very general. The application can specify boolean operations (AND, OR, NOTAND), ranges, and partial matches as well as the indices to be searched.

A Xiphias CD-ROM product may use a number of separate databases to hold data for different subsystems. However, the interface to each of these is identical. The application need only be concerned with linking the data returned to the data displayed.

Databases can be transfered in binary form between Macintosh and Amiga systems. The export format can be used to transfer to and from the MS-DOS environment.

The system consists of seven separate libraries of C language functions. These are arranged in a hierarchical manner. The application generally accesses only the top level library directly.

Xiphias Database System

The top level of the system is called the Xiphias Data Base, or XDB, layer. It provides a database management system for Xiphias products.

Schema file

Each database of an application has a "schema" that determines the data fields, indices, sort keys, etc for that set of data. This is passed to the XDB_SE system as a single string. It can be stored as part of the application or as a text file.

Database file layout

Each database will consist of:

- ☐ a single primary ISAM file,
- ☐ a secondary title file: single string for each story made up of fields specified by the schema. It is kept separately for fast retrieval to give the user a summary of hits from a search. Two types of title files are supported: one for fixed length titles, one for variable length titles.
- ☐ a collection of inversion index files
- ☐ an optional threads file

All will have similar base names, extensions will vary to distinguish files. Extensions can be any length by cannot contain white space. The exact number and names of the files is determined by the schema file. PC file name limitations are not enforced for all systems. However the 8+3 MS-DOS limitations may be a consideration to cross-platform compatibility.

Threads and Save-search files

It is often desireable to save the results of a search for later re-use. The XDB system provides two methods for saving search results: Save-search files and threads. Save-search files are an ASCII text file that holds the record ids and titles for a search. The name of the file must be known to recover the search. Threads provide a more powerful mechanism as they are kept in an indexed file. Each thread consists of a name and an array of record IDs. The name is indexed and can contain any characters. The array of record IDs is the same structure returned by a search.

Tools are provided that automatically create thread or save-search files using either prefix searches or a list of search strings. Threads created in this manner can be used to speed up large searches by first checking if a thread exists for a search string. If the thread exists, it can be loaded and used instead of executing the standard search. If no thread exists, the standard search is executed and only a minimum time delay will be suffered.

Alternatively, an application may provide a means for interactively creating the record ID list and saving it in either a thread or a Save-search file. These editorially created lists can be used for "guided tours" of the hypertext.

Primary data structures

Database Handle. Once a database is opened, it is referenced by a Database Handle. This is only needed when closing or changing the active database.

Story Data. The application and the XDB system pass a linked list of field data when new data is added or retrieved.

Found Array. When a search is conducted, the system returns an array of story IDs that match the search criterion. The first element of the array specifies the number of hits.

Titles of Found Array. A routine is provided to obtain the titles for each item in the found array in a special structure. The maximum size of the title structure can be specified by the application.

Inversion Indices. The schema defines a number of index trees and indices within these trees. A tree may hold any number of indices. A field can be inverted into zero or more indices. An index may receive data from more than one field.

The fields can be inverted with several methods: one word, phrase, phrase stem, and full inversion. One word inversion puts the entire field as one entry in the index. Phrase inversion puts each line of the field separately into the index, without removing spaces. Phrase stemming is similar to phrase, but certain common plurals are reduced to their stem before adding. Full inversion puts each word longer than two characters in the field into the index separately.

All methods reduce the strings to lower case. A stopword list can be used to automatically eliminate common words from the full inversion. Inversion has a special method to deal with hyphenated words. Such words are indexed three times. Once as a whole word without the hyphen (dash or slash), once for the words on each side of the hyphen. This allows the stings F-16, F/16 and F16 to be equivalent.

Interface functions

Open/Close Database. These functions handle opening and closing of database files. The Open/Create functions accept a string representation of the schema. Close uses the database handle.

Set Database. There can be many open databases at a time, however only one is specified as the "Current Database". Functions are provided to switch databases.

Get Story Functions (First/Last/Next/Prev/ID). A set of functions to get the first/last/next/specific record ID are provided.

Search. The search function takes an expression string and a list of indices to be searched. It returns an array of record IDs that satisfy the expression. The expression the form:

```
<expression>->  <factor>
              | <factor> && <factor>          //AND
              | <factor> || <factor>          // OR
              | <factor> &! <factor>          // AND-NOT

<factor> ->      ( <expression> )             // Grouping
              | [ <indexList> ] <factor>
              | <word>*                       // Prefix Search
              | <word>:<word>                 // Range Search
              | <word>                        // simple search

<word>   ->     <string of alphanumeric chars>
```

For example:

```
airplane
```

searches all evaluation indices for the word "airplane"

```
[words] airplane
```

searches the index "words" for the word "airplane"

```
[words] airplane |  [words] airplanes
```

searches the index "words" for the word "airplane", searches the index "words" for the word "airplanes" then does a logical OR on the two arrays.

```
[words] airplan*
```

searches the index "words" for the words with the prefix string "airplan"

```
[words] airplane && [dates] (1945:1990)
```

searches the index "words" for the word "airplane", then searches index "dates" for stories from 1945 to 1990, Then it does a logical AND of the two lists.

Get Titles. This function returns a structure holding all the titles for records referenced in an array of ids. The maximum size of this structure is specified by the application. The function will fill the structure as far as possible and return. The application must check that the number of titles is equal to the number of hits. A mismatch indicates insufficient memory to hold all the titles.

Add/Modify/Delete. Functions are provided to add/modify and delete stories from the database. Note that modify and delete do not remove the old inverted references. These are retained until the database is condensed. Old records (deleted or modified) can be retrieved if necessary.

ISAM Layer

An ISAM file manager library is provided. This is used to store the data for the stories. A record can have multiple instances of a field. Fields in a record can be of arbitrary length. Records currently have a maximum length of approximately 64K due to I/O constraints.

Functions are provided for manipulation of fields and records.

RecFile Layer

The ISAM system is built on a layer called RecFile. This layer provides indexed sequential access to records of arbitrary length. The primary difference is that ISAM provides for interpretation of Fields within the record. The RecFile simply stores and returns a chunk of bytes.

Btree Layer

The Btree layer provides a proprietary Btree system used for both the inversion indices and the RecFile/ISAM record management. The Btree is capable of storing multiple numeric and string indices in the same file. Several methods for sorting strings are supported. This is necessary if numbers are to be stored as strings, i.e., for large date ranges, etc. Numbers do not sort properly according to standard ASCII sort order.

Cache Layer

A Block File cache layer is used by the Btree to hold recently used nodes. The size of the cache can be set by the application. A single shared cache or separate caches can be used.

Link List Layer

A general purpose doubly linked list structure is defined with a set of functions to support it. The design of this structure and library is such that the application can define its own Node structures that will be compatible with the library.

Xiphias Portability Library

Xiphias has developed a library that allows application programs to be ported between Macintosh, Amiga, MS-DOS and Windows 3.0 platforms. It defines a layer of memory management, message functions and general purpose tools to hide the differences between the platforms. (NOTE: this library does *not* address the user interface portability problems. That is far beyond the scope needed to support XDB/SE).

Application shell

Each platform has different requirements for User Interface, etc. Different application shells are written for each platform (i.e. Hypercard and XCMDS are used on the Macintosh). Some code is shared between products on the same platform. Non-Xiphias applications must develop their own shell system for their platforms.

Auxiliary Tools

Xiphias has a number of auxiliary database tools that support the creation of its CD-ROM products.

***Converting Schemas*.** Often the production database has more fields, different sorting order, or other data that is not needed in the product system. A tool, XDBcopy, is provided that can convert data from one schema to a corresponding schema. This is used in particular to replace a sort index with ISAM story ordering. It is also used to compact the database by removing deleted and old version of modified records and their inversion data.

***Btree Compaction*.** The index files are stored using a proprietary Btree structure. This is efficient for production systems which may do modifications to the data. However, the tree will by design contain a fairly large amount of unused space. A tool, BtCopy, has been written that can compress out most of the unused space.

***Misc Tools*.** Other tools are available to examine the contents and gather statistics on database and index files. Tools are provided for the importing and exporting of data into a standard text format that can be ported between platforms.

A more detailed document describing all the libraries and their entry points is available upon request from your local Commodore techsupport person.

Time Fold Compression System

Time Fold (hereafter TFold) is a video and animation compression system. TFold is designed to work on animations previously created and stored as files in the standard ANIM format. After these animations are brought into TFold they are processed and compressed, which reduces their size by up to sixfold. The processing results in a slight degradation in picture quality. Fine detailing may be restored to selected areas of each cel by the artist/user, through TFold's stenciling feature. Depending on how much detail is required, this process can be automated.

In return for this degradation in image quality, you receive an increase in frame rate. The minimum frame rate must be fast enough to create the illusion of motion. The lowest rate accepted in the industry for animation is eight cels per second. For video, 10-12 cels per second is more suitable. If the artist/user using TFold follows a few simple guidelines, a 12 cel per second rate is guaranteed. This allows enough extra bandwidth for full digital stereo sound. Although at this time there is no support for audio in TFold the connection of video and audio is a trivial task.

The files TFold creates are geared toward being displayed on a standard composite monitor of RF-based television. The image processing that TFold performs on an animation takes this into consideration, and this affects the fianl look of the video or animation images. These images will not look as good on an RGB monitor.

After the artist/user has worked through his/her ANIM and created a TFold format file, the programmer may play the file back by utilising the "Tfold.o" object module that is linkable with both Lattice and Manx "C" standard object modules. The size of TFold.o is approximately 5 Kbytes.

TFold accepts animations in 3 formats: ANIM Opt 5, ANIM Opt 5 Xor, and TFold. The first two are standard formats, used by numerous Amiga tools. The TFold format is specific for TFold, and contains compressed files.

A brief list of features of TFold:

- ☐ TFold allows users to concatenate two animations together.
- ☐ TFold and TPlay let you play back animations from both memory and disk. Users can create and play back animations that exceed the memory capacity of the system.
- ☐ Stencil drawing. TFold lets the user/artist create his own stencils, on a cel-by-cel basis, or globally for the entire animation. For example, imagine a series of video frames displaying a person moving about a room and talking. The user can create a stencil over the face of the figure. When TFold compresses the animation, it does no compression on the face, but only on the rest of the cel. Thus the face remains clear, while the rest of the image may be blurred or otherwise degraded by the compression.
- ☐ Dynamic size data. TFold calculates and displays the size of the current cels, previous cel, and next cel, making it easy to assure that cels are not too large.

CDXL Toolkit

Purpose

CDXL is a technique which permits rapid transfer of data from the CD-ROM to the appropriate area of memory in the CDTV player. The results can be impressive. CDXL can be used for transferring any type of data: code, indexes, IFF files, etc. Yet one of the most frequently proposed applications for CDXL is the creation of short "video sequences".

These sequences, as have been demonstrated, can display an image in 1/4 screen, 6 bitplanes, at 12 frames per second, including mono audio playback digitized at approximately 8 Mhz. If compression techniques are used on the data, the image size, the frame rate, or both may be increased.

In order to facilitate the creation and manipulation of these video sequences, Commodore has developed a set of CDXL tools. The first version of these tools should be available when you read this. These are CLI-based utilities. Future enhancements might include a GUI-based interface.

These tools permit the creation of "CDXL files". These files are a series of video images, including sound, which have been stripped of all unnecessary header information. It is important to minimize the amount header information, as any unnecessary data will slow down the frame rate.

Data Capture

These CDXL tools work on IFF image and 8SVX sound files. They do not handle the capture of those files. That must be done with existing Amiga-based tools.

Video capture is usually best performed by framegrabbers. The following method has been used successfully to capture video sequences:

☐ Transfer the video source to a laser disc. This can be done either with write-once laserdisc systems, or by sending your source tape to a laserdisc pressing house. This step, while not absolutely necessary, has certain advantages. It assures proper frame alignment , and avoids burn-out of the heads on VCR equipment, caused by intensive use of the pause and frame-advance functions.

☐ Using AmigaVision or a similar tool, write an ARexx script to control the video capture of the framegrabber board. Your script basically controls the laser disc player, telling it to advance to the next frame, pause, trigger a grab on the framegrabber, save the file to disk, and begin again.

☐ Scaling of the image may be done with tools, such as The Art Department Professional. Again, an ARexx script can be written to automate this procedure.

Audio capture may be performed with existing audio capture tools and editors.

CDXL Toolkit Description

All these tools will use a standard CDXL file header. CDXL files that do not contain this header cannot be processed. A sample playback routine, including source code, will be provided for CDTV developers to use.

Here folows short descriptions of each of the tools.

☐ ***XLMake.*** Creates a CDXL file from separarte IFF images.

XLMake is the primary tool for creating CDXL image files. The purpose of this tool is to accept standard IFF files that are already in the desired resolution and color depth, translate them into a raw bitplane format, and combine them together into a single CDXL data sequence. Header information containing details about resolution, colors, and audio samples is inserted as each frame is built.

This tool is normally used by adding it to the script file or ARexx program used to capture individual video frames; thus, it works in conjunction with existing Amiga tools like the Art Department Professional. Each time a new frame has been captured, scaled, and color adapted, XLMake is invoked to combine the frame to the CDXL file under construction. To help illustrate its use, a useful ARexx/ADPRO script will be provided.

☐ ***XLShow.*** Shows a CDXL file on a stock Amiga 2500/3000

XLShow provides a means of viewing CDXL video sequences on a standard Amiga computer. The purpose of this tool is to allow developers to view their CDXL files at any time in order to make decisions and changes without mastering a CD (or creating an emulator run).

This tool attempts to simulate CDXL playback as it would occur on a CDTV. Although the general timing characteristics of playback are preserved (total play time), the speed of the Amiga File System and lack of CDXL on standard Amigas prevents precise timing from being achieved. The resulting playback is only an approximation.

☐ ***XLInfo.*** Displays information about a CDXL file

XLInfo displays detailed information about a CDXL file. The primary purpose of this tool is to provide accurate timing and other information for application developers. This tool will calculate and display the following information:

total play time
number of frames
frame resolutions
number of colors
color modes
audio track size
audio period estimate
sequencing information
out of sequence frame numbers

A "Quick" option saves time by providing limited information (playtime, approximate number of frames, audio track size) without needing to read the entire file (measurement is based on first frame).

☐ ***XL Join.*** Splices together two or more CDXL files

XLJoin is a quick way to combine multiple CDXL files into a single video sequence. The purpose of this tool is to provide application developers with a way of splicing together separate video scenes to create a single CDXL video file.

XLJoin also allows the insertion of CDXL files anywhere within another CDXL file. This tool does not alter its input files in any way, thus preserving them for use in other video sequences.

☐ ***XLTrim.*** Removes frames from a CDXL file

XLTrim supplies a simple set of editing functions for CDXL files. Its purpose is to provide a reliable means of removing selected frames from CDXL files. This is often necessary to remove undesired video, reduce the play time, or speed-up the action.

The editing features are controlled with CLI command arguments. Options are provided to:

> trim video frames from the beginning, middle, or end.
> remove a frame every N frames.
> trim the entire file down to a specified number of frames or playback time.

This program also has the capability of removing the audio track from a CDXL file (useful when remastering audio).

☐ ***XLCopy.*** Copy a sequence of frames to a new CDXL file

XLCopy copies frames from any point in a CDXL file and transfers them to a new file. The purpose of this tool is to provide a simple means of extracting useful video sequences without altering the original file. This approach is easier and safer than XLTrim in some situations.

☐ ***XLAudio.*** Inserts an audio track into a CDXL file

XLAudio adds an audio sound track to an existing CDXL file. This is normally done as the last step in creating a CDXL audio/video sequence. The audio must already be adjusted to the correct sampling rate (using tools like AudioMaster, etc.) before laying down the track. The tool will expand the Audio track on each CDXL frame if necessary, or overwrite the track if one already exists. The sound track can be mono or stereo.

Note that the performance of these tools will vary depending on the Amiga processor and hard disk. Due to the speed of the Amiga File System, large CDXL files may require significant time to process.

CDTV Tools Disks

Commodore currently provides 2 disks to licenced CDTV developers. One diskette, called the ISO DevPak diskette, contains all the files necessary to create an ISO image of your AmigaDOS application. Refer to the notes on PreMastering and Mastering for more information.

The second diskette, called CDTV Tools, contains tools useful for CDTV development. Here is a brief summary of the contents of that tools diskette:

Directory	**Contents**
CDXL	Sample source code demonstrating the use of CDTV transfer lists.
Bookit	The Bookit utility, intended for use in your startup-sequence, reads in and activates the CDTV Preferences settings. This utility, included in the ISO DevPak vers. 3.5, replaces the CDTVPrefs utility. See the file bookit.readme for more information.
Debox	The latest version of the SBox and Debox data compression routines.
Fonts	Two sample fonts, which you may use in any CDTV application on a royalty-free basis.
Includes	The latest versions of the include files for CDTV.
Keeper	A utility, intended for use in your startup-sequence, which loads and displays any IFF image while the rest of your program is loading. This should help avoid long delays with nothing for the user to look at during loading.
CD-DA	Sample source code which plays CD-DA tracks on a CD.
Audio	Routines to play large 8SVX samples (greater than 128K), and conversion utilities to work with files imported from other platforms.
Playerprefs	Contains the autodocs for the playerprefs.library. This library in the ROM of the CDTV contains numerous routines for reading and writing to the Bookmark device, calling the system screensaver, interpreting joystick events as mouse events, etc.
Memcard	Code to read and write to the personal memory card, including information on how to make the card bootable.
System	System level utilities to read Preferences, dump a sector, source code showing how to write to bookmark memory, and how to stop resets when a disc is ejected.
LHArc	Utility to compress and decompress files. Includes documentation.
NTSC-PAL	Utilities for switching between NTSC and PAL screens.

Authoring Systems and Languages

Numerous authoring systems exist for the Amiga. Some of these have been used successfully with CDTV.

The Director II

This new version of the Director, from the Right Answers Group, a script-based system, has numerous improvements that make it one of the best available tools for creating CDTV applications.

The CDTV module is Copyright 1991, Right Answers, Inc. A license agreement must be negotiated with Right Answers, Inc. before it can be distributed in a commercial product. The module makes use of special libraries and devices that are contained within the CDTV ROM, and will not necessarily operate on a standard Amiga system.

Here is a list of the CDTV-specific extensions the Right Answers Group has added to The Director II:

PLAYMSF:

```
cdtv result,"playmsf",startm,starts,startf,endm,ends,endf
cdtv {variable},"playmsf",<expr>,<expr>,<expr>,<expr>,<expr>,<expr>
```

Plays the CD audio in MSF format (minutes,seconds,frames) from the "start" to "end" times specified.

PLAYTRACK:

```
cdtv result,"playtrack",startt,starti,endt,endi
cdtv {variable},"playtrack",<expr>,<expr>,<expr>,<expr>
```

Play a CD audio track, starting from track 'startt' index 'starti' and ending on track 'endt' index 'endi'. If endt and endi are 0, it will stop at the end of the current track.

PLAYLSN:

```
cdtv result,"playlsn",sector,count
cdtv {variable},"playlsn",<expr>,<expr>
```

Play CD audio starting from logical sector number 'sector' for 'count' sectors.

PAUSE:

```
cdtv "pause",mode
cdtv "pause",<expr>
```

Enter PAUSE mode if mode=1, leave PAUSE mode if mode=0.

CHECK:

```
cdtv result,"check"
cdtv <variable>,"check"
```

Check if play has completed, will return a "result" variable of 1 if completed, a 0 if not.

WAIT:

```
cdtv "wait"
cdtv "wait"
```

Wait for audio play to complete.

STOPPLAY:

```
cdtv "stopplay"
cdtv "stopplay"
```

Terminate the audio play.

FADE:

```
cdtv "fade",v,c
cdtv "fade",<expr>,<expr>
```

Fade the volume level to volume ’v’ (where 0 is off and 32767 is full) in ’c’/75ths seconds.

FRAMECOUNT:

```
cdtv result,"frame"
cdtv <variable>,"frame"
```

Return "result" set to current CD frame count.

FRONTPANEL:

```
cdtv "frontpanel",set
cdtv "frontpanel",<expr>
```

Enable frontpanel if "set" = 1, disable if ’set’ = 0. This may not work to enable the front panel after a PLAY command has been issued.

QUICKSTATUS:

```
cdtv result,"quickstatus"
cdtv <variable>,"quickstatus"
```

Set "result" to current status.

CHGINT:

```
cdtv result,"chgint",val
cdtv <variable>,"chgint",<expr>
```

Will remove disk change interrupt, so that disk can be changed without reboot. val=0 remove interrupt, val=1 add interrupt. UNTESTED.

KILL:

```
cdtv "kill"
```

Abort the CDTV module.

MOVEM:

```
cdtv "movem",x,y
cdtv "movem",<expr>,<expr>
```

Allows you to explicitly set the position of the mouse pointer. This can be useful when using the CDTV remote in the mouse mode. You can set the mouse (with the pointer display off) to a position in the middle of the screen, and once the mouse moves, determine which direction it moved and reset it to the middle of the screen for the next move, so that the mouse will never get to the edge of the screen and be unable to move further in that direction.

LOAD:

```
cdtv "load",buff,name
cdtv "load",<expr>,<string>
```

Load a Squeezebox compressed image named "name" into The Director image buffer "buff". The image is maintained as a compressed image, and cannot be displayed directly. Use the CDTV "COPY" command to decompress.

RESOLUTION:

```
cdtv "resolution",buff,array
 cdtv "load",<expr>,<array>
```

Will set the first 3 locations in "array" to be the x,y and depth values of the compressed image in buffer "buff". The depth will return 4096 if it is a HAM image. The "array" parameter must be a 2 byte per element array and be at least 3 elements long. (DIM res[3],2)

COPY:

```
cdtv "copy",srcbuff,dstbuff
cdtv "copy",<expr>,<expr>
```

Decompresses the Squeezebox compressed image in buffer "srcbuff" into image buffer "dstbuff". Dstbuff must exist and be of the proper resolution or an error will result. Use The Director's NEW command to create a buffer to decompress into. Note that color cycling information is not preserved in the current implementation. The RANGE command can be used to explicitly set cycling of the uncompressed image.

AMOS Language

AMOS Basic, from Mandarin Software, is a sophisticated development language with more than 500 different commands. It can be used to create arcade games, adventure games, educational programs, video titling sequences, etc.

AMOS Basic contains a built-in animation Language, called AMAL. AMAL programs are executed 50 times a second, using an interrupt system. A compiler option is now available, as well as a runtime module.

Here are a list of some features:

☐ Define and animate hardware and software sprites (bobs) quickly and easily

☐ display up to eight screens at once, each with its own colour palette and resolution (including HAM, half-brite and dual playfield modes)

☐ scroll a screen easily. Create multi-level parallax scrolling by overlapping different screens.

☐ Use AMAL to create complex animation sequences for sprites, bobs, or screens which run on interrupts

☐ Play Soundtrack, Sonix, or GMC tunes or IFF samples on interrupt

☐ Use commands like RAINBOW and COPPER MOVE to create color bars

AmigaVision

AmigaVision can be used for CDTV applications. It is an excellent prototyping tool.

However, AmigaVision has a major stumbling block: the size of the run-time system (approximately 230 Kbytes) is such that creating large applications that run on a 1 Mbyte system can be difficult. A flow (or program) that has minimal complexity, with more than 10 or 12 instructions, can often exceed the available RAM.

A new version of AmigaVision is being developed that allows chaining of flows. One flow can load and run another, thus avoiding overly large flows.

◆

Debugging Amiga Software

by Carolyn Scheppner

This article presents some general techniques for debugging software on the Amiga. Before you start programming the Amiga, it's a good idea to read the development guidelines in the introductions of the the Addison-Wesley Hardware Manual (ISBN 0-201-18157-6) and ROM Kernel Reference Manual: Libraries and Devices (ISBN 0-201-18187-8). These guidelines contain important rules which are applicable to all Amiga programs, configurations, and operating system releases. Additional information can be found in the Troubleshooting Guide published in Amiga Mail and in the Libraries and Devices manual. These documents cover the most common Amiga programming problems.

Preventing Bugs

The best way to debug software is to prevent bugs in the first place. Accordingly, here are seven basic rules you should always follow when writing Amiga software:

1. Use *Enforcer* and *MungWall* while developing your code.
2. Read the latest autodocs and the include file comments for the functions and structures you are using.
3. Always check return values from system functions. Provide a clean way out and useful messages if something fails! Assembler programmers - remember to `TST.L D0` (or `MOVE.L D0` into memory or a non-address register) after system calls, before branching on condition codes.
4. C Programmers - Use function prototypes for system functions and your own functions. It's a little extra work but will save you time in the long run by immediately catching most types of improper function calls (missing arguments, swapped arguments, etc.)
5. Keep a version number in your code and update the version number whenever changes are made. The 2.0 VERSION command can print out the version of any executable which contains a specially formatted version string:

   In C: `UBYTE *vers="\0$VER: programname 36.10";`
   In Asm: `vers DC.B 0,'$VER: programname 36.10',0`
6. Document the code changes for each version. This can be done manually or by using a document control system such as RCS.
7. Test your code! Test on different configurations, under low memory and error conditions, and in conjunction with various watchdog tools. Test your product with *MungWall*, if possible in conjunction with *Enforcer* to catch uses of null pointers and freed memory.

Finding and Fixing Bugs

It is hard to generalize about debugging because different kinds of bugs often require very different approaches. A bug report from a user is quite different from a bug that you've just introduced in new code! However, all debugging requires some common steps:

1. Define the problem
2. Narrow the search and find the bug
3. Understand, and fix the bug
4. Make sure you didn't just break something else

Steps 3 and 4 are the same for all types of bugs, so we'll cover those last. Steps 1 and 2 require different approaches for different kinds of bugs. Here are some examples.

A.You've added or written new code and something is broken.

1. *Define the problem.* Make sure you can reproduce the problem so you'll know when it's gone. Define as "when I do xxx the program does (or doesn't do) do yyy."

2. *Narrow the Search.* If you just added a couple of lines of code, and have the same development environment as before, check your source code first. Check for misuse of existing variables, improper error checking, improper use of system or internal functions, and possible changes to conditional program flow.

If you can't spot the problem, it's time to slow it down and see what's going on. Use a source-level or symbolic debugger, or print/kprint/dprint debugging, with delays added if necessary. One particularly useful type of debugging statement is:

```
printf("About to do xxx.  k =%ld Ptr1=$%lx...\n",k,Ptr1);
Delay(50);
```

The delay gives the debugging line time to be output and gives you a chance to read it before the action is taken. See mydebug.h on the 1991 DevCon disks for easy ways to add conditional debugging statements like this to your code.

By stepping through or printing out your actions and variables, you will generally be able to isolate the bug. If you have isolated the area but still can't find the bug, re-read the autodocs for the routines you are using. Check the Troubleshooting Guide in the Addison-Wesley *Libraries and Devices* manual. Check all other uses of the variables in the problem area. If all else fails, isolate the problem code by writing the smallest possible example that demonstrates the problem.

If the problem is not present in the smallest possible example, then go back and check your code. If the problem is still present, contact CATS for assistance or upload the example to BIX (note - one of the quickest ways to find bugs in a small source code example is to upload the source to BIX amiga.dev/main and ask what's wrong with it).

B. Your code has intermittent problems that you can't pin down, or appears to trash something under certain conditions.

1. *Define the problem.* It is difficult to reproduce intermittent problems, so try to force the problem to show itself. First try running your program with *Enforcer* and *MungWall*. If you don't have an MMU, use *WatchMem* and *MungWall*, but be prepared to crash a lot. If you don't get any hits, try the same thing during low-memory situations, heavy multitasking and device IO, etc. If you are doing Exec device IO, try *IO_Torture* to catch premature reuse of IORequests. Hopefully, you will pick up a hit.

2. *Narrow the Search.* If you have no *MungWall/Enforcer* hits, try some debugging statements or source-level debugging to follow the values of your variables. Use *TStat* to see if your stack usage is high. Check all possible areas where you might be overwriting the end of an array or otherwise trashing memory. Re-read the Autodocs for the system functions you are using.

If you are reusing an IORequest too soon, check your source code (debugging would just slow down your execution and might give the IORequest a chance to complete, masking the problem).

If you have *Enforcer* hits, use debugging statements or a debugger to step through your code WHILE running *MungWall* and an MMU watchdog tool (or *WatchMem*). This will allow you to pinpoint where the problem occurs.

C. Your code works fine on one system but not on another. Or you've received a bug report from a user.

1. *Define the problem.* First find out the exact configuration of the system the problem occurred on. Important elements include memory configuration and addresses, amount of free Chip and Fast RAM, processor type, custom chip version, expansion peripherals, OS version, and other software in use when the problem occurred. The *Config* program on the 1991 DevCon disks is useful for printing out much of this information.

The memory address ranges can be particularly important now that machines are available with memory beyond the 24-bit address limit. For example, overwriting a byte array by one byte now has a good chance of trashing a 32-bit address variable, or even your routine's return address on the stack.

If a user reports the problem, find out the exact version of your software they are running, how they launched the program, and what their stack is set to (if launched from CLI). Try to get them to reproduce the problem in a known environment (ie. after booting with a release Workbench diskette). Get their phone number and keep it with a record of all of the information you can get on the problem. Keep bug reports in an organized form. If you get two reports on the same problem, you can be pretty sure that the problem really exists, and the combined information may help you track it down.

2. *Narrow the Search.* Attempt to reproduce the problem. If you can't reproduce it immediately, try stepping through the problem area while using *Enforcer* and *MungWall.* If you don't get any hits, try again with less memory available and other tasks running. Try to reproduce the user's configuration and environment. If you still can not reproduce the problem, ask the user to come up with a simple repeatable sequence which causes the problem on a system booted with a normal release Workbench disk.

Read the Troubleshooting guide in the Addison-Wesley *Libraries and Devices* manual or *Amiga Mail* for information on the causes for many problems that only show up in certain configurations or environments.

If all else fails, look carefully at your code for misuse of variables or system functions, and for improper error-checking or cleanup after any allocation or open. Check that all cleanups are done in the proper order.

D. Your program loses memory.

1. *Define the problem.* First make sure that you are actually losing memory. Use *Flush* (from the 1991 DevCon disks) and Avail to check for actual memory loss.

Set up your system so you have a shell window available and can start your program without moving any windows (re-arranging windows causes memory fluctuations). Test for memory loss as follows. First, try *Flush* and *Avail* a few times to make sure nothing else in your system is causing memory to fluctuate. Then perform the following steps.

1. *Flush*
2. *Avail* (write down the Fast, Chip, and total memory free)
3. Start your program and use its features
4. Exit your program
5. *Flush*
6. *Avail* (compare the Fast, Chip, and total free to previous figures)
7. If you have a loss, go back to step 2.

Testing such as shown above will flush out all properly closed devices, libraries, and fonts which have been loaded from disk by your program and other programs. This allows you to check for actual memory loss.

Note that under 2.0, since the audio.device is ROM-resident but not initialized by the system until it is opened by someone, the first program to use the audio device or speech capabilities will appear to cause a small but permanent memory loss. This is the memory allocated for the audio device's base structures. If your program uses audio or speech, first use the SAY program or SPEAK: before performing the above memory loss test so that the audio device's initial memory usage will not interfere with your tests.

One special memory loss problems is a continual loss of memory while a program is running. This is generally caused by not keeping up with IntuiMessages, or not freeing Locks.

2. *Narrow the search.* Try the above test again, but this time just start your program and exit immediately. If you do not lose memory, try several times more, using some of your program's features, and attempt to determine which part of your program causes the memory loss. Check your source code for all opens and allocations and check for matching frees and closes, in the proper order, for each of them.

The size of a memory loss can also be a clue to the cause. For example, a loss of exactly 24 bytes is probably a Lock() which has not been UnLock()'d. Knowing the exact size of the loss (as determined with *Flush* and *Avail*) is important when you try determine which allocation is not being freed.

Some additional tools on the 1991 DevCon disks can help determine where memory losses occur. You can use *MemMon* to record the relative memory usage as you test various parts of your program. *Snoop* can be used to record all memory allocations and frees on a remote terminal, after which *SnoopStrip* can strip out all matching pairs. *MungWall* contains an enhanced snoop option for tracking only the memory allocations of a particular task. *MemList*, which outputs the system memory list, can also be useful when debugging memory loss and fragmentation.

The *Wedge* program, which can restrict its reporting to the function calls made by a single task or list of tasks, can also be used to monitor the allocations and frees done by your task. By inserting debugging statements, you can mix status messages ("About to do xxx") with *Wedge* or *MungWall* SNOOP output. Examine the output for an allocation which matches the size of your loss. Use LVO's WEDGELINE option to generate command lines for *Wedge*.

Removing Bugs

I mentioned steps 3 and 4 earlier. This is the easy part (finding the bug is the hard part). These steps are the same for most debugging problems.

3. *Understand, and Fix the Bug.* When you find the bug, make sure you understand it. Don't just try something else. If you are having a problem with a system routine, read the autodocs and chapter text for that routine. Check the Troubleshooting Guide in the 1.3 Addison- Wesley *Libraries and Devices* manual.

When you understand what is wrong, fix the problem, being especially careful not to affect the behavior of any other parts of your program. Carefully document the changes that you make and bump the revision number of the program. Note your changes in the initial comments of the program, and in the area where the changes were made.

4. *Make Sure You Didn't Break Anything New.* Try to reproduce the problem several times and make sure it is gone. Thoroughly test the rest of your program and make sure that nothing else has been broken by your fix. Test your program in combination with watchdog tools such as *MungWall* and *Enforcer*.

Debugging Tools

1. Software Watchdog and Stressing Tools

The MMU-based Amiga debugging tool *Enforcer* provides debugging and quality assurance capabilities far beyond what was previously possible. It is now possible to find bugs even in code that appears to be working perfectly - the kinds of bugs that could cause serious problems on different configurations. *Enforcer* is able to trap improper low memory accesses, writes to ROM, and accesses of non-existent memory - problems which are generally caused by use of freed or improperly initialized pointers or structures.

When used in conjunction with a free memory invalidation tool such as *MungWall*, additional illegal memory uses are forced out into areas trappable by *Enforcer*.

Another extremely useful testing tool, especially for assembler programmers, is *Scratch* by Bill Hawes. One of the most surprising compatibility problems we have seen is improper use or dependence on scratch registers (D1,A0,A1) after a system call. *Scratch* allows you to invalidate the contents of these scratch registers after system calls so that improper usage of these registers in your code may be brought out.

It is also useful to test your software with stressing tools such as *EatMem*, *Memoration*, and *EatCycles*, in conjunction with *Enforcer* because *Enforcer* will help to catch use of unsuccessful allocations immediately.

All software should be tested with these tools during development, and should be required to pass a test with *Enforcer* in conjunction with *MungWall*, *Scratcher*, and *IO_Torture* before being released and distributed.

2. Symbolic and source-level debuggers

Symbolic debuggers allow you to trace and single step through your code, and examine or change your variables and structures. The source-level debuggers which are provided with some compilers allow you to trace and single step your code at the source-level after compiling with special flags. Debuggers can often be used in combination with other tools such as *MungWall* and *Enforcer* to detect exactly where a problem is occurring.

3. Printf() and kprintf() / dprintf() debugging

This simple method of debugging allows you to monitor where you are, what your variables contain, and anything else you care to print out. Printf debugging is suitable for any process code that is not in a Forbid or Disable (printf breaks a Forbid or Disable). Kprintf (serial) and dprintf (parallel) debugging is more flexible and can be used in process, task, or interrupt code. The kprintf function is provided in the debug.lib linker library. The parallel version, dprintf, is provided in the ddebug.lib linker library. See the debug.lib kprintf Autodocs for more information on the types of formats handled by kprintf and dprintf.

Kprintf outputs to the serial port at whatever baud rate the port is currently set to. Generally, kprintf is done at 9600 baud with a terminal, or another Amiga running a terminal package, connected to your serial port with a null modem serial cable.

However, it is possible to kprintf to yourself (ie. to a terminal package running on your own machine) if you have a modem attached to your serial port, and your terminal package set to the baud rate of your modem. Obviously, if the problem you are debugging causes you to crash, a remote terminal is a better choice. The ASCII capture feature of your terminal package can be used to capture the kprintf debugging output for later examination.

Remote (kprintf/dprintf) debugging is extremely useful when combined with other remote debugging tools such as *Enforcer* and *IO_Torture* because your own debugging statements will be interspersed with the remote output of the other debugging tools, allowing you to track what your program is doing when problems occur.

Printf/kprintf/dprint debugging can be conditionally coded more conveniently by using an include file such as mydebug.h (see the DevCon disks). Mydebug.h eliminates the need for messy `#ifdef` and `#endif` lines around your debugging statements by providing the conditional macros D(bug()), D2(bug()), and DQ(bug()) which take printf-style format strings and arguments in their inner parens. One handy feature of these macros is that your debugging statements can be be quickly changed from printf's to kprintf's or dprintf's by simply setting a flag in mydebug.h and recompiling.

Example: `D(bug("I'm here now and a=%ld",a));`

4. Other ways to debug low -level code

If you can't link with debug.lib, low level code can also be debugged by inserting visual or audio cues to let you know where you are. DebTones.asm (in *AmigaMail* and on the 1991 DevCon disks) demonstrates a small audio tone macro suitable for debugging low level code. Another common method is flashing the power LED (see togl_led.asm), or doing an Intuition DisplayBeep() to flash the screen.

5. Specialized debugging tools

A variety of specialized debugging tools are available for monitoring and debugging such things as system function calls, device IO, process status, memory usage, and software errors. These tools can be used without recompiling your program and can provide valuable debugging information. See the list of tools accompanying this article in the DevCon notes.

How to Use MMU Watchdogs and Other Remote Debugging Tools

Remote Serial Debugging

Hardware Setup: When hooking two Amigas together, use a straight RS-232 cable with a null-modem adaptor, or use a null-modem cable. When hooking an Amiga up to another type of computer or terminal, you may or may not need the null-modem (crossed lines) depending on whether the other machine's RS-232 port is designed to be basically a sender or a receiver. Avoid connecting lines which are not directly related to RS-232 because different computers have various power supplies and grounds on these other lines. My null modem debugging cable is wired as follows:

Amiga	Terminal
1	1
2	2
3	3
7	7
5, 6, 8	5, 6, 8
20	20

Software: For remote debugging at 9600 baud, set the sending machine's Preferences to 9600 baud, and use a 9600 baud terminal or an Amiga running a 9600 baud terminal package (preferably with ASCII capture capability) as the receiving machine. Note that other baud rates can also be used for most serial debugging because normal serial kprintf's do not modify the serial SERPER register and are therefore output at the last baud rate your serial hardware was set to. Test your setup by copying a small text file to SER: or try the *ktest* program from the 1900 DevCon disks. The output should show up on the remote terminal.

Applications can output serial debugging statements by using kprintf from C or KPrintF from assembler and linking with amiga.lib and debug.lib. See the debug.lib Autodocs for more information. Serial input functions are also available. Also see the mydebug.h conditional debugging macros on the 1991 DevCon disks.

Serial Watchdog software setup: Make sure your test machine is set to the same baud rate as the remote terminal you are connected to. Turn on the ASCII capture of your remote terminal.

For machines with 68030 or 68020+MMU:

```
[RUN] MungWall      (removable with CTRL-C, or BREAK n if RUN)
[RUN] IO_Torture
Enforcer ON
```

For non-MMU machines (warning - encourages bad software to crash!)

```
[RUN] MungWall      (removable with CTRL-C, or BREAK n if RUN)
[RUN] IO_Torture
[RUN] WatchMem
```

Setup for Local Serial Debugging

Hardware: If you have a modem attached to your serial port, it is possible to capture your own serial debugging output locally. This setup can be useful as long as the problem you are debugging is not one which crashes the machine.

Software: Run a terminal package at your modem's baud rate to capture the kprintfs. You probably won't be able to test this setup by copying a file to SER: (since the terminal package probably has an exclusive open on the serial device). Instead, use a small program like *ktest* (on the 1991 DevCon disks) to test your setup, or, if you already have an MMU watchdog installed, try an illegal memory accessor such as *Lawbreaker*. Use the terminal package's
ASCII capture feature to capture your debugging output.

Serial satchdog software setup: Same as for remote serial debugging, but first start up a terminal package on the test machine, at the baud rate of the attached modem, with ASCII capture turned on.

Setup for Parallel Debugging

Hardware: To set up for parallel debugging, attach a parallel printer to the Amiga's parallel port and turn the printer on. Note - if no device is attached to the parallel port, parallel debugging statements will hang waiting for the port hardware.

Software: Some debugging commands have options for parallel rather than serial output. Examples include *Enforcer.par, MungWall.par, IO_Torture.par,* and *Wedge* with the 'p' option. Also, your can send your own debugging statements to the parallel port by using dprintf from C, or DPutFmt from assembler, and linking with ddebug.lib and amiga.lib. On the 1991 DevCon disks, see dtest.asm for an example of calling DPutFmt from assembler, and mydebug.h for debugging macros which can use printf, kprintf, or dprintf.

Parallel atchdog software setup:

For machines with 68030 or 68020+MMU:
[RUN] MungWall.par (removable with CTRL-C, or BREAK n if RUN)
[RUN] IO_Torture.par
Enforcer.par ON

For non-MMU machines (warning - encourages bad software to crash!)
[RUN] MungWall.par (removable with CTRL-C, or BREAK n if RUN)
[RUN] IO_Torture.par
[RUN] WatchMem

◆

AmigaGuide

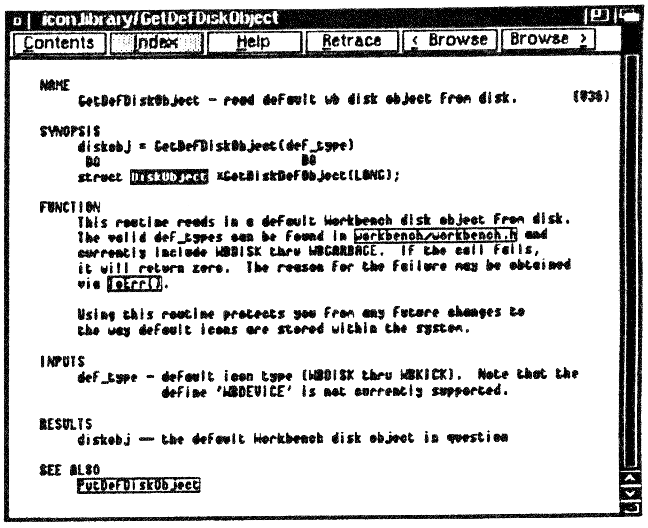
by David N. Junod

The standard Amiga keyboard sports a HELP key, yet there has been no system provided support for this key. Now, there is *AmigaGuide*, which provides a standard method of displaying help and other on-line documentation to the user.

Capabilities

AmigaGuide uses an Intuition window that contains a scroll bar, buttons and pull-down menus, to display plain ASCII text files or *AmigaGuide* databases.

An *AmigaGuide* database is a set of related documents contained in one file. Each document may contain references to other documents, using what is called a link. A document may contain any number of links, pointing to any number of other documents. When the user selects a link, the document that the link points to will be displayed. The user may then use the links to read through the database, following whatever path they may choose. The technical term for *AmigaGuide*'s abilities, is hypertext.



```
NAME
     GetDefDiskObject - read default wb disk object from disk.      (V36)

SYNOPSIS
     diskobj = GetDefDiskObject(def_type)
      D0                        D0
     struct DiskObject *GetDiskDefObject(LONG);

FUNCTION
     This routine reads in a default Workbench disk object from disk.
     The valid def_types can be found in workbench/workbench.h and
     currently include WBDISK thru WBGARBAGE.  If the call fails,
     it will return zero.  The reason for the failure may be obtained
     via IoErr().

     Using this routine protects you from any future changes to
     the way default icons are stored within the system.

INPUTS
     def_type - default icon type (WBDISK thru WBKICK).  Note that the
                define 'WBDEVICE' is not currently supported.

RESULTS
     diskobj — the default Workbench disk object in question

SEE ALSO
     PutDefDiskObject
```

Figure 1 - AmigaGuide

The user may at any time print a document or a portion of the document. They may also send portions of a document to the clipboard, for use in other applications.

Using ARexx, the user may write scripts, or an application could provide scripts, to control *AmigaGuide*.

Cross-reference tables can be loaded that specify where a keyword, or phrase is defined. The user can then use *AmigaGuide*'s Find Document facility to quickly display a document based on keyword, without having to know the name of the database that it is located in.

AmigaGuide provides a unique feature to hypertext systems, called Dynamic Nodes. A Dynamic Node is a hypertext or plain text document that is generated in real-time as opposed to coming from a static file. An application that generates Dynamic Nodes is called a Dynamic Node Host.

Interfacing

AmigaGuide databases are accessed in three different ways:

- Databases can be browsed directly from the Workbench or Shell using a utility named *AmigaGuide*, a hypertext replacement for *More*.
- *AmigaGuide* support can be added to an existing application, that supports ARexx, by using *AmigaGuide*'s ARexx function host capabilities.
- Applications can use the functions of *AmigaGuide* to provide help on gadgets, menus and windows. For example, the user could position the pointer over any gadget or menu item, press help, and the appropriate document would be displayed in the *AmigaGuide* window. The application could also have *AmigaGuide* display a pertinent portion of the current project.

Other Uses

In addition to help or on-line documentation, *AmigaGuide* has other possible uses. For example,

Tutorials - An application that has an ARexx port and supports *AmigaGuide* could set up a help system that not only provides help, but could also give examples. The user could read about a feature, then click on the EXAMPLE button, which would run an ARexx script that would give an example of use. For instances, to show Pattern Fill, the script could draw a circle, select a pattern, and then fill the circle.

Computer Aided Instruction - The student could read about different topics, following links. A multiple choice quiz could be set up at the end where the questions and answers run ARexx scripts to accumulate the score.

Program by Query - Many programmers develop using a Cut & Paste technique. They clip modules from the various applications or utilities that they have written, and paste them together to build new applications. A database could be set up of all these different code fragments (such as loading and saving ILBM's, clipboard access, playing sounds, etc.), you could step through, answering questions, while the sections that you need are being appended to a new source file.

Using *AmigaGuide* from Workbench or CLI

A number of utilities are provided to access *AmigaGuide* databases from the Workbench or Shell. These utilities are:

AmigaGuide - This tool is much like the system utility, *More*, but is also capable of interpreting *AmigaGuide* databases.
LoadXRef - Used to load a cross-reference from disk into memory. Multiple files can be loaded.
ExpungeXRef - Used to clear the cross-reference table from memory.

AmigaGuide

The *AmigaGuide* utility is used for browsing through *AmigaGuide* databases. It can be run from the Workbench or the Shell.

To use it from Workbench, set the default tool of an *AmigaGuide* database's icon to *AmigaGuide*. Currently, the *AmigaGuide* utility doesn't recognize any tool type parameters.

From the Shell, *AmigaGuide* uses the following command template.

```
DataBase,Document/K,Line/N,PubScreen/K,PortName/K:
```

The following are descriptions of the arguments:

DataBase - The name of the *AmigaGuide* database to display. *AmigaGuide* will look in the current directory for the database. If the database isn't found, then *AmigaGuide* will search through its path for the database. To set the path, see the Path paragraph in the ''User Preferences'' section.

Document - The document within the database to display. *AmigaGuide* will use the cross-reference table to automatically supply the DataBase and Line parameters.

Line - The line of the document to start displaying from.

PubScreen - The public screen to open on. Remember that public screen names are case sensitive.

PortName - The name to assign to the ARexx port for this occurence of *AmigaGuide*.

LoadXRef

The *LoadXRef* utility can be used from Workbench or the Shell to load cross-reference files from disk into memory. Multiple cross- reference files may be loaded at a time.

From Workbench, just set the default tool of a cross-reference file's project icon to *LoadXRef*. This tool doesn't support any tool type parameters.

From the Shell, just specify the name of the cross-reference file to load. *LoadXRef* will look in the current directory for the file. If the file isn't found, then *LoadXRef* will search through the users preference path for the file.

ExpungeXRef

The *ExpungeXRef* utility is used to remove all entries from the cross-reference table in memory.

User Preferences

AmigaGuide allows a number of items to be tailored to the users' preference. These preference items are stored in environment variables. The AmigaDOS command *SetEnv* can be used to set any of these variables.

A Preference Editor that sets the *AmigaGuide* preferences would write in the *ENV:AmigaGuide* directory when "Use" is selected, and write in the *ENVARC:AmigaGuide* directory when "Save" is selected.

The following is a list of the variable names, and what they control.

Path - This variable contains the list of directory names that *AmigaGuide* will search through when it attempts to open a database. The directory names are separated by a space. For example:

```
SetEnv AmigaGuide/Path "Workbench:Autodocs Workbench:Includes"
```

Text - Used to specify the graphical style that the links are presented in. The possible styles are:

BUTTON	Draw a raised border around the text (default).
UNDERLINE	Underline the text.
BOLD	Bold the text.
ITALIC	Italicize the text.

For example:

```
SetEnv AmigaGuide/Text BUTTON
```

Pens - This variable provides the user with the ability to specify the colors to use for the various renderings that *AmigaGuide* performs.

```
SetEnv AmigaGuide/Pens <abcdefgh>
```

Where:

a = Background pen
b = Button text pen
c = Button background pen
d = Highlighted button text pen
e = Highlighted button background pen
f = Outline pen
g = Highlight outline pen
h = Text on background pen

For example:

```
SetEnv AmigaGuide/Pens 21213001
SetEnv AmigaGuide/Text BOLD
```

Internally, *AmigaGuide* subtracts '0' from the pen number, so values can range from 0 to 207.

Authoring AmigaGuide Documents

Authoring an *AmigaGuide* database, or any hypertext database for that matter, is a difficult task. It takes a lot of insight into the subject matter and how the pieces relate to each other. A database must consist of documents that relate to each, documents must be broken up into manageable chunks, and links must be carefully thought out. A document should consist of information dealing with one topic. Each document should contain links to other related documents.

An *AmigaGuide* database is ASCII text with embedded commands that tell *AmigaGuide* how to interpret the database. A database should consist of a main table of contents and a number of related documents.

Label Commands

These are commands that can be used within a database. Commands must start in the very first column of a line. If a line begins with an @ sign, then it is interpreted as a command.

`@DATABASE <name>`
Must be the very first line of a Amiga HyperText document.

`@MASTER <path>`
Complete path of the source document used to define this HyperText database.

`@NODE <name> <title>`
Indicate the start of a node (page/article/section). The first node, or main node, must be named MAIN. MAIN must be the master table of contents for the database.

`@DNODE <name>`
Indicates the start of a dynamic node. The HyperText system uses the callback hooks to obtain the document from a document provider.

`@WIDTH <chars>`
How wide, in characters, the largest document is.

`@HEIGHT <chars>`
How high, in characters, the largest document is.

`## <remark>`
`@REMARK <remark>`
Remark (not displayed to the user).

Node Label Commands

These are commands that can be used within a @NODE.

`@ENDNODE <name>`
Indicate the end of a node. Must start at the beginning of a line.

`@TITLE <title>`
Title to display in the title bar of the window during the display of this node. Must start at the beginning of a line.

`@TOC <node name>`
Name of the node that contains the table of contents for this node. Defaults to MAIN. This is the node that is displayed when the user presses the "Contents" button.

`@PREV <node name>`
Node to display when the user selects "< Browse"

`@NEXT <node name>`
Node to display when the user selects "Browse >"

`@{<label> <command>}`
Indicate a textual link point. Can be anywhere in a line.

Action Commands

These are commands that can be assigned to a link point.

`ALINK <name> <line>`
Load the named node into a new window, with <line> at the top of the display.

`CLOSE`
Close the window (should only be used on windows that where started with alink).

`LINK <name> <line>`
Load the named node, with <line> at the top of the display.

`RX <command>`
Execute an ARexx macro.

`RXS <command>`
Execute an ARexx string file. To display a picture, use 'ADDRESS COMMAND DISPLAY <picture name>', to display a text file 'ADDRESS COMMAND MORE <doc>'.

`SYSTEM <command>`
Execute an AmigaDOS command.

`QUIT`
Shutdown the current database.

Example *AmigaGuide* Database

The following is an example of an *AmigaGuide* database. It doesn't contain any 'useful' information, but it does show the usage of some of the commands.

```
@database "example.guide"
@master "example.doc"

@node Main "Example AmigaGuide database"

Table of Contents

@{"ARexx" link ARexx}
@{"Shell" link Shell}
@{"Workbench" link Workbench}

@endnode

@node ARexx
Put something here about ARexx
@endnode

@node Shell
Put something here about the Shell
@endnode

@node Workbench
Put something here about Workbench. Say that it has @{"icons" link icon}.
@endnode

@icon "Workbench Icons"
Those little pictures that you can drag around.
@endnode
```

ARexx Scripts

It is possible to control *AmigaGuide* using ARexx. Each occurrence of *AmigaGuide* has an ARexx port. The *AmigaGuide* shared system library is also an ARexx function host.

Port Naming

The default port name is AMIGAGUIDE.# where # is the occurrence. With the *AmigaGuide* utility, a port name can be specified as a command line argument. An application with an *AmigaGuide* interface can also provide the port name.

ARexx Commands

Any of the following action commands are also ARexx commands. All commands are not case-sensitive.

`ALINK <name> <line>`
Load the named node into a new window, with <line> at the top of the display.

`CLOSE`
Close the window (should only be used on windows that where started with alink).

`LINK <name> <line>`
Load the named node, with <line> at the top of the display.

`SYSTEM <command>`
Execute an AmigaDOS command.

`QUIT`
Shutdown the current database.

ARexx Functions

The *amigaguide.library* is an ARexx function library. The library can be added as a function host with the following lines:

```
/* Load the HyperText library as a function host */
IF ~SHOW('L','amigaguide.library') THEN
CALL ADDLIB('amigaguide.library',0,-30)
```

It supports the following functions (function names are not case-sensitive).

`ShowNode`
PUBSCREEN/K,DATABASE/K,NODE/K,LINE/N
Display a node on the named screen. Defaults to the Main node, on the Workbench screen. If DATABASE isn't specified, then will search through the cross-reference list to get the database name.

`LoadXRef`
NAME/K
Load a cross-reference file into memory.

`GetXRef`
NODE/K
Return information on NODE. Format of the text string returned is "NODE" "DATABASE" TYPE LINE.

`ExpungeXRef`

Flush the cross-reference list from memory.

Adding An AmigaGuide Interface To Your Application

Applications can add *AmigaGuide* support, using the functions within the *amigaguide.library*.

The HyperApp example on disk illustrates how to add context sensitive help to an application.

Cross Reference Files

AmigaGuide allows cross-reference tables to be loaded that specify what document a keyword is defined in. This cross-reference table is used by the ''Find Document'' requester to locate a node. It is also used by the *AD2HT* utility to construct hypertext versions of the system Autodoc files.

A cross-reference file follows a layout similar to the *devs:mountlist format*. The table itself starts with a line that consists of the keyword XREF: and ends with a line that contains a # as the only uncommented character. Comments can be included in the C-style format, beginning with "/*" and ending with "*/".

```
/* This is a comment */
XREF:
..
"Gadget"     "intuition/intuition.h"215 3
...
#
```

A cross-reference entry consists of four words:

Keyword - The keyword that is being defined.
File - The ASCII file or database that the keyword is defined in.
Line - The line within the node that the keyword is defined on.
Type - This field indicates the type of keyword. Possible values are:

0 Generic *AmigaGuide* link.
1 Describes a function.
2 Describes a command.
3 Points to an include file.
4 Describes a macro.
5 Describes a structure.
6 Describes a structure field.
7 Describes a type definition.
8 Describes a define.

Loading a Cross Reference List

A cross-reference list can be loaded from disk using the LoadXRef() function. The format is:

```
success = LoadXRef(lock, name);

LONG success;
BPTR lock;
STRPTR name;
```

The arguments are:

lock - Lock on the directory where the file is located. May be NULL.
name - Name of the cross-reference file to load. *LoadXRef* will search the user preference path.

Returns:

-1 Indicates that the load was aborted by a control-C.
0 Unable to load the file.
1 Successfully loaded the file.
2 No changes have been made since the last time that this file was loaded.

Access to the Cross Reference List

An application can use the GetAmigaGuideAttr() function to obtain a pointer to the cross-reference list. The application then may search through the list, or even save the list to disk. Note that access to this list is read-only, and must be enclosed between a call to LockAmigaGuideBase() and UnlockAmigaGuideBase().

```
struct List *list;
LONG key;

/* Lock the AmigaGuideBase for exclusive access */
key = LockAmigaGuideBase(NULL);

/* Get a pointer to the cross-reference list */
if (GetAmigaGuideAttr(AGA_XRefList, NULL, &list))
{
 /* Do something with the list */
}

/* Unlock AmigaGuideBase */
UnlockAmigaGuideBase(key);
```

The cross-reference list consist of nodes of struct XRef, defined in *<libraries/amigaguide.h>*.

```
/* Cross-reference node */
struct XRef
{
    struct Node xr_Node;/* Embedded node */
    UWORD xr_Pad;  /* Padding */
    struct DocFile *xr_DF;/* (Private) Document defined in */
    STRPTR xr_File;/* Name of document file */
    STRPTR xr_Name;/* Name of item */
    LONG xr_Line;  /* Line defined at */
};

#defineXRSIZE      (sizeof (struct XRef))
```

The following are the field definitions:

xr_Node - Embedded node structure. xr_Node.ln_Name points to xr_Name. xr_Node.ln_Type contains the type of the keyword.
xr_Pad - Used to align the remaining fields.
xr_DF - Private pointer.
xr_File - Pointer to the name of the file that xr_Name is defined in.
xr_Name - Pointer to the keyword.
xr_Line - The line, within xr_File, that xr_Name is defined on.

Dynamic Node Host

AmigaGuide provides a unique feature to hypertext systems, called Dynamic Nodes. A Dynamic Node is a hypertext or plain text document that is generated in real-time as opposed to coming from a static file. An application that generates Dynamic Nodes is called a Dynamic Node Host.

If a link point within a document isn't resolved, it will query a list of Dynamic Node Hosts to see if any one of these external applications can resolve the node.

This feature allows for dynamic interaction with constantly changing data. This feature is useful for *AmigaGuide* authoring tools, interactive development environments, extremely context sensitive help systems, to name a few.

Dynamic Nodes has been implemented using an Object Oriented Programming paradigm. When a link point hasn't been resolved a HM_FindNode message is sent to each Dynamic Node Host on the list. Once the node has been found, then a HM_OpenNode is sent to the Dynamic Node Host that the node belongs to. HM_CloseNode is sent to the host once the node has been exited.

Initializing a Dynamic Node Host

In order for an application to register itself as a Dynamic Node Host, it must initialize a hook and add the hook to the *AmigaGuide* Dynamic Node list, using the AddAmigaGuideHost() library call.

The hook structure as defined in <*utility/hooks.h*>:

```
/* Standard hook structure */
struct Hook
{
    struct MinNode h_MinNode;
    ULONG (*h_Entry)();/* assembler entry point */
    ULONG (*h_SubEntry)();/* often HLL entry point */
    VOID *h_Data;  /* owner specific */
};
```

The AddAmigaGuideHost() function returns a pointer to an AmigaGuideHost structure. This structure, defined in <libraries/*amigaguide*.h>, is as follows:

```
/* Callback handle */
struct AmigaGuideHost
{
    struct Hook agh_Dispatcher;/* Dispatcher */
    ULONG agh_Reserved;/* Must be 0 */
    ULONG agh_Flags;
    ULONG agh_UseCnt;/* Number of open nodes */
    APTR agh_SystemData;/* Reserved for system use */
    APTR agh_UserData;/* Anything you want... */
};
```

Following are the field definitions for the AmigaGuideHost structure.

agh_Dispatcher - This is a copy of the Hook that was passed to AddAmigaGuideHost().
agh_UserData - Can be manipulated by the Dynamic Node Host any way it sees fit.

The other fields are not to be manipulated in any way.

Removing a Dynamic Node Host

A Dynamic Node Host is removed using the RemoveAmigaGuideHost() library function. The application must successfully remove the hook before exiting, otherwise *AmigaGuide* would end up calling the hook function, that has been unloaded from the system, causing a system crash.

The following code fragment illustrates how to initialize and remove a Dynamic Node Host.

```
#include <exec/types.h>
#include <libraries/amigaguide.h>
#include <clib/exec_protos.h>
#include <clib/amigaguide_protos.h>
#include <pragmas/exec_pragmas.h>
#include <pragmas/amigaguide_pragmas.h>
#include <stdio.h>

extern struct Library *SysBase, *DOSBase;
struct Library *AmigaGuideBase;

#define     ASM     __asm
#define     REG(x)  register __ ## x

ULONG __saveds dispatchDNH(struct Hook *, STRPTR, Msg);
ULONG ASM hookEntry(REG(a0) struct Hook *,REG(a2) VOID *,REG(a1) VOID *);

/* Callback hook dispatcher */
ULONG __asm hookEntry (
            REG(a0) struct Hook *h,
            REG(a2) VOID *obj,
            REG(a1) VOID *msg)
{
/* Pass the parameters on the stack */
    return ((h->h_SubEntry)(h, obj, msg));
}

main (int argc, char **argv)
{
    struct Hook hook;
    AMIGAGUIDEHOST hh;

    /* amigaguide.library works with 1.3 and newer versions of the OS */
    if (AmigaGuideBase = OpenLibrary ("amigaguide.library", 33))
    {
        /* Initialize the hook */
        hook.h_Entry = hookEntry;
        hook.h_SubEntry = dispatchDNH;

        /* Add the AmigaGuideHost to the system */
        if (hh = AddAmigaGuideHost (&hook, "ExampleHost", NULL))
        {
            printf ("Added AmigaGuideHost 0x%lx\n", hh);

            /* Wait until we're told to quit */
            Wait (SIGBREAKF_CTRL_C);

            printf ("Remove AmigaGuideHost 0x%lx", hh);

            /* Try removing the host */
            while (RemoveAmigaGuideHost (hh, NULL) > 0)
            {
                /* Wait a while */
                printf (".");
                Delay (250);
            }
            printf ("\n");
        }
        else
        {
            printf ("Couldn't add AmigaGuideHost\n");
        }

        /* close the library */
        CloseLibrary (AmigaGuideBase);
    }
}
```

Handling Dynamic Node Host Messages

Once the Dynamic Node Host has been added to *AmigaGuide*, it can start receiving messages for different requests.

Currently, *AmigaGuide* supports the following methods, or message types, for a Dynamic Node Host.

HM_FindNode - When *AmigaGuide* can't resolve a link, then it sends a HM_FindNode message to all Dynamic Node Hosts to see which host defines the node.

HM_OpenNode - Once *AmigaGuide* locates the host that defines a node, using the HM_FindNode message, then the HM_OpenNode message is sent to that host to ask it to open the node.

HM_CloseNode - Once the user has closed all occurrences of a Dynamic Node, then *AmigaGuide* sends the HM_CloseNode message to the host that opened the node.

HM_Expunge - *AmigaGuide* sends this message to all Dynamic Node Hosts when the Expunge vector of *amigaguide.library* is invoked, or the ExpungeDataBases() function is called.

Several of the methods receive a TagItem array as an argument. Currently the following tags are supported. The tag values are defined in <libraries/*amigaguide*.h>.

HTNA_Screen - A pointer to the screen on which source *AmigaGuide* window resides.

HTNA_Pens - The pen array associated with the screen.

HTNA_Rectangle - A Rectangle structure (defined in <graphics/gfx.h>) containing the dimensions of the window.

Each method requires one or more parameters. The MethodID is the only common parameter for each method.

HM_FindNode

Used to locate the Dynamic Node Host that a node is defined by. When a Dynamic Node Host receives a HM_FindNode message for a node that it owns, it should reply with TRUE, otherwise it must respond with FALSE.

The HM_FindNode method receives the following arguments:

```
/* HM_FindNode */
struct opFindHost
{
    ULONG MethodID;
    struct TagItem *ofh_Attrs;/*  R: Additional attributes */
    STRPTR ofh_Node;/*  R: Name of node */
    STRPTR ofh_TOC;/*  W: Table of Contents */
    STRPTR ofh_Title;/*  W: Title to give to the node */
    STRPTR ofh_Next;/*  W: Next node to browse to */
    STRPTR ofh_Prev;/*  W: Previous node to browse to */
};
```

The field definitions are as follows:

ofh_Attrs - This field contains a pointer to a TagItem array of attributes for the message. This field is read-only.

ofh_Node - The name of the node to open. This field is read-only. It is possible for this name to contain parameters that need to be parsed. For example, the command that triggered the link could have been:

Link "snd/beep 320"

In which case, the ofh_Node field would contain:

beep 320

ofh_TOC - The Table of Contents to assign to this node. This is the name of the node to link to, if the "Contents" button is pressed. This field can be written to (not implemented).

ofh_Title - The title to assigned to this node. This field can be written to.

ofh_Next - The name of the logical next node. This is the name of the node to link to if the "Browse >" button is pressed. This field can be written to.

ofh_Prev - The name of the logical previous node. This is the name of the node to link to if the "< Browse" button is pressed. This field can be written to.

HM_OpenNode

Once *AmigaGuide* locates the host that defines a node, using the HM_FindNode message, then the HM_OpenNode message is sent to that host to ask it to open the node. If the Dynamic Node Host is able to open the node, then it should respond with TRUE, otherwise respond with FALSE.

The HM_OpenNode method receives the following arguments:

```
/* HM_OpenNode, HM_CloseNode */
struct opNodeIO
{
    ULONG MethodID;
    struct TagItem *onm_Attrs;/*  R: Additional attributes */
    STRPTR onm_Node;/*  R: Node name and arguments */
    STRPTR onm_FileName;/*  W: File name buffer */
    STRPTR onm_DocBuffer;/*  W: Node buffer */
    ULONG onm_BuffLen;/*  W: Size of buffer */
    ULONG onm_Flags;/* RW: Control flags */
};
```

The field definitions are as follows:

onm_Attrs - This field contains a pointer to a TagItem array of attributes for the message. This field is read-only.

onm_Node - The name of the node to open. This field is read-only. It is possible for this name to contain parameters that need to be parsed. For example, the command that triggered the link could have been:

Link "snd/beep 320"

In which case, the onm_Node field would contain:

beep 320

onm_FileName - If you want *AmigaGuide* to read a particular node from disk, then supply the file name here. The file can either be a straight ASCII file or an *AmigaGuide* document (not a database). The application can write to this field.

onm_DocBuffer - If you are dynamically creating a node in memory, then use this field to point to the buffer. If this field is used, then the onm_BuffLen field must be filled in also. The application is in charge of freeing onm_DocBuffer when it is done (indicated by a HM_CloseNode message). The application can write to this field.

onm_BuffLen - The length of the buffer that onm_DocBuffer points to. The application can write to this field.

onm_Flags - These are control flags that the Dynamic Node Host can set.

HTNF_KEEP - Don't flush this node from memory until the database is closed. This will delay the HM_CloseNode message until the database is closed.

HTNF_ASCII - The node is straight ASCII, doesn't contain any *AmigaGuide* keywords.

HTNF_CLEAN - Remove the node from the database as soon as it is closed.

HTNF_DONE - This flag is used to indicate to *AmigaGuide* that the Dynamic Node Host already took care of presenting the node, and that there is no need for *AmigaGuide* to present it. This is useful for playing sounds, animations, or even debugging information.

HM_CloseNode

Once the user has closed all occurrences of a Dynamic Node, then *AmigaGuide* sends the HM_CloseNode message to the host that opened the node. If the Dynamic Node Host is able to close the node, then respond with TRUE, otherwise respond with FALSE.

The HM_CloseNode message uses the same message structure as HM_OpenNode.

HM_Expunge

AmigaGuide sends this message to all Dynamic Node Hosts when the Expunge vector of *amigaguide.library* is invoked, or the ExpungeDataBases() function is called. The Dynamic Node Host should free as much memory as it possibly can.

The HM_Expunge method receives the following arguments:

```
/* HM_Expunge */
struct opExpungeNode
{
    ULONG MethodID;
    struct TagItem *oen_Attrs;/*  R: Additional attributes */
};
```

The field definition is as follows:

oen_Attrs - Currently, no attributes passed.

Message Dispatcher

The following is an example of Dynamic Node Host message dispatcher. Since this is executed with a callback hook, it is being run on the calling process' task, not the Dynamic Node Host process. Because of that, it is necessary to load the global data segment using geta4() or __saveds.

```
ULONG __saveds
dispatchAmigaGuideHost (struct Hook *h, STRPTR db, Msg msg)
{
    struct opNodeIO *onm = (struct opNodeIO *) msg;
    ULONG retval = 0;

    switch (msg->MethodID)
    {
        /* Does this node belong to you? */
        case HM_FindNode:
            {
                struct opFindHost *ofh = (struct opFindHost *) msg;

                DB (kprintf ("Find [%s] in %s\n", ofh->ofh_Node, db));

                /* See if they want to find our table of contents */
                if ((stricmp (ofh->ofh_Node, "main")) == 0)
                {
                    /* Return TRUE to indicate that it's your node,
                     * otherwise return FALSE. */
                    retval = TRUE;
                }
                else
                {
                    /* Display the name of the node */
                    Display (onm);

                    /* Return TRUE to indicate that it's your node,
                     * otherwise return FALSE. */
                    retval = FALSE;
                }
            }
            break;
        /* Open a node. */
        case HM_OpenNode:
            DB (kprintf ("Open [%s] in %s\n", onm->onm_Node, db));

            /* See if they want to display our table of contents */
            if ((stricmp (onm->onm_Node, "main")) == 0)
            {
                /* Provide the contents of the node */
                onm->onm_DocBuffer = TEMP_NODE;
                onm->onm_BuffLen = strlen (TEMP_NODE);
            }
            else
            {
                /* Display the name of the node */
                Display (onm);

                /* Indicate that we want the node removed from our
                 * database, and that we handled the display of the
                 * node */
                onm->onm_Flags |= (HTNF_CLEAN | HTNF_DONE);
            }

            /* Indicate that we were able to open the node */
            retval = TRUE;
            break;

        /* Close a node that has no users. */
        case HM_CloseNode:
            DB (kprintf ("Close [%s] in %s\n", onm->onm_Node, db));

             /* Indicate that we were able to close the node */
            retval = TRUE;
            break;

        /* Free any extra memory */
        case HM_Expunge:
            DB (kprintf ("Expunge [%s]\n", db));
            break;

        default:
            DB (kprintf ("Unknown method %ld\n", msg->MethodID));
            break;
    }

    return (retval);
}
```

Developer Specific Utilities

On the DevCon disks are a number of utilities for *AmigaGuide* that would be of special interest to the developer.

AD2HT - Scans Autodocs for function and command names, scans the INCLUDE: directory for .h include files, and scans the include files for structure definitions. Also parses Autodoc files and constructs a corresponding *AmigaGuide* database. It resolves links to functions, commands, include files and structures.

CXRef - Scans a source directory and builds a cross-reference showing where all functions and structures are defined. Something like ctags on the Sun.

Glossary

Autodoc - Documentation extracted from source code.

browse - Navigate sequentially through a series of documents, instead of via links.

cross-reference table - A table that consists of the following information:

keyword - A word or phrase

database - Name of the database that the keyword is defined in.

line - The line that the keyword is defined on within the database. This only applies if the database is a straight text file (such as an include file).

type - What type the keyword is, such as a 'normal', function, command, include file, or structure.

database - A file that consists of multiple documents.

document - A block of text, constrained to one subject. Also called a node.

Dynamic Node - A Dynamic Node is a hypertext, or plain text, document that is generated in real-time, or from live data, as opposed to coming from a static file.

Dynamic Node Host - An application that generates Dynamic Nodes.

link - A word, or phrase, within a document that is linked to another document.

node - A block of text, constrained to one subject. Also called a document.

retrace - To follow, in a reverse direction, the path taken through a series of documents.

table of contents - A list of documents, categorized by type.

Recommended Reading

"Hypertext Hands-On!" Ben Shneiderman & Greg Kearsley. Addison-Wesley Publishing Company. ISBN 0-201-13546-9

"Understanding Hypertext Concepts and Applications" Philip Seyer. Windcrest Books. ISBN 0-8306-9108-1 (hard) 0-8306-3308-1 (pbk.)

* * *

CDTV Audio Cookbook

Notes from the Front Line

by Jim Hawkins

What follows is a brief account of the main factors involved in producing a successful Mixed Mode CD. It comes from some very late nights and some very inelegant programming on the Music Maker project. It is not definitive. Every programmer will have a specific need and a specifically economical route to it. All code here is in pseudo-code, because Amiga programs are now written in variety of languages--C, assembler, Amos, Modula-2, and (in my case now) Oberon (the first efficient Object Oriented language available on the Amiga). I am going to assume that programmers are familiar with the use of Device drivers on the Amiga. If they are not, they'd better go away and practice before committing their product to a very expensive circle of plastic...

Audio Play

A Mixed Mode CD has up to 99 ''tracks'' on it. The word track here is really only a logical division of the total CD data into sub-sections. Although the structure of sectors in a data track is different from that in a CD-DA track, there is no need to give it even a second's thought.

On a Mixed Mode disk track ONE is always data - probably an AmigaDOS image sitting inside the standard CD-ROM file system known as ISO9660. Playing track ONE on an audio player can produce anything from an impromptu KraftWerk experience to speaker cones vanishing down the street. The other tracks can of course be anything from a few seconds of musical stab to a complete Mahler symphony. But effectively, the disk space can always be seen in terms of time. Usually, a CD contains 72 minutes of time. That time can then be subdivided into data time and audio time. Taking a rough calculation that data gets eaten at the rate of 160K per second, a minute's worth of time=approximately 60*16000 bytes, or crudely 9,600,000 bytes. Experience shows that it is safer to err on the generous side (CD disks have a couple of minutes silence at the start and some at the end.) So a good, easy, decimal number is 10 megabytes per minute. Supposing, therefore, that you have 53 minutes of audio; you can then see that that will leave you about 190 megabytes for your data track. That's roughly the way it is on Music Maker--although the data track is in fact about 100 megabytes, leaving 90 megabytes unused, or 9 minutes if we think in terms of time.

So the first cook-book rule is: work out the ratio of data to audio. The more audio the less data, and vice versa.

Playing the audio tracks you have on the disk is easy. Except, of course, you will not be able to do that until you have a disk. So you will be in the uncomfortable position for programmers of not being able to test it until it's too late! So unless your sponsors have lots of money and patience it's a seriously good idea to do protective coding. More on that later.

There are two (or more, who knows?) ways of playing CD-DA tracks using the *cdtv.device*. One is a simple PlayTrack call and one is PlayMSF. For most purposes PlayTrack is the best call to use because it doesn't matter if your track wanders about the disk in various iterations of the development loop. If it's track 8 it will always be track 8. You will need to use the MSF format only very very precise start times. The PlayTrack example in the CDTV Developers' notes gives a perfect example. To summarize the steps:

1. Get a Port
2. Get a StdIO structure
3. Open the *cdtv.device* on the StdIO
4. Fill in the struct:
    a) ioCommand <- PLAYTRACK
    b) ioLength <- 0
    c) ioData <- NIL
5. Either DoIO() or SendIO() this StdIO request.

Now comes the interesting bit, and from experience talking to lots of developers with problems, the most complex issue facing you on CDTV, so I've given it a section on its own.... think of it as an interrupt.

Terminating Conditions

The audio track will play until it finishes if you don't stop it. There are two ways of stopping a track... you either AbortIO() the request, or you issue a STOPTRACK command and then WaitIO() the intial request. This is all fine and dandy, because you have issued the command. But consider the case where you have been off during a SendIO() doing other things, wazzing copper lists and playing sampled sounds. You might assume that the track has finished when it hasn't, or that it is still running when it isn't. This will be bad news if you try to open or read from a file!

So - the method is to use CheckIO, and be absolutely certain that the track has really really stopped before doing anything else. If you don't, and the audio is still playing, you are likely to lock the CDTV up, causing upset and general refusal to pay invoices.

This is what I call protective coding. *Check* that you have closed any files *before* playing the audio (there could be a file read still pending) and *check* that you have stopped the track before doing anything involving the CD audio or file-system. It's a very easy trap to get into, I've been there and done it, and it has serious consequences.

If you take those steps, playing audio tracks is easy--providing that the track exists, of course.

Track Checking

Normally you will know how many tracks you have. But suppose you have a condition where the user can pop the CDTV disk and stick an ordinary CD-DA disk into the drive. Two calls to the device are relevant now. IS_ROM and TOC. The TOC structure with a call to track ZERO tells you the first and last track, from which you can limit the number of tracks the user can attempt to play. It's also a good idea to use an IS_ROM call. This can alert you to the fact that track ONE is a data track and can eat loudspeakers. It's arguably their own fault if they start popping disks during normal program run when that's outside the rules of the game, and I would not personally bother with doing endless numbers of TOC calls unless the popping of disks is intrinsic, as it is in Music Maker - DrumAlong. Remember also that the CD-DA side of the system and the file-system side are sharing the same drive, so if they DO pop the disk you'd better check that your disk is back in
before even thinking about opening a file.

Crude method for doing this: if you now do a TOC call and find that IS_ROM returns TRUE and the disk has the right number of tracks, assume it's the right one.

More sophisticated version: do as above, and check your manufacturer's ID and product ID. Bet you most times it will be unnecessary.

PLAYMSF

All you are doing with a PlayMSF call is playing from a time to a time. This is where you face the GREAT DILEMMA... where are the tracks???

To keep the software update and test cycle to the minimum you must work closely with the CD manufacturers. There is a big difference between mastering what are essentially Amiga programs on a CD and mixed mode disks. If you are going to synchronize anything to an audio track it is VITAL that the tracks don't drift between cycles in the development. And they can. Go for a good manufacturer, because they will create a master tape that leaves the tracks (after initial editing (see later comments)) starting and stopping at almost exactly the same frame. Ask for your disk to be laid out such that the audio tracks are at the end, leaving a large data track space into which your updated data and program images will be placed. If you don't do this, you will suffer from wandering track syndrome, which can be expensive.

For many applications it is appropriate to obtain a CD with the audio tracks placed at the end and nothing in track one, which could be several minutes of silence. In other words, if you leave a big enough hole for your data you can go through many iterations of replacing the data track whilst leaving your audio in exactly the same physical space on the disk.

You can now find out where your tracks are by (a) reading the TOC (b) getting a print-out from the manufacturer. This gives you a crude idea. You can then mess with your program (or, to be more sensible, a control file) and set an exact time to start each track from. You are now independent of the disk layout, lead-ins, any silence from the transfer from DAT or analogue tape. You are also crucially vulnerable to wandering tracks.

The MSF format in the *cdtv.h* file looks a bit obscure with all those shifts in it, but what it boils down to is simple: a LONG (32 bit) word with the 3 LS bytes holding M S F; or to put it in a civilized language:

```
TYPE MSF=RECORD
dummy,minutes,seconds,frames:BYTE
END;
```

The dummy is there simply to force the right-justification (and there have been nights when sucking a dummy seemed the only way out of the problems!). As usual, the *cdtv.h* information is simple and accurate about this. Set up the StdIO request, SendIO() or DoIO() it, and enjoy the speed at which the track comes up.

Nothing in life is simple, so there are a few potential nasties here. As it says in the *cdtv.h* file, JUST ABORTING a PlayMSF call is not enough, because the light stays on and the laser is still active. Here is an example where you MUST use CD_STOPTRACK even if the track has in fact stopped. Unless you want to pretend you're doing full motion video of course, but then, who isn't? (pretending)

How Do I Know When It's Started?

The CD seek time can be up to 0.8 secs, and the laser has to position itself and all that technical stuff. I'll probably get a dozen rude faxes from Carl Sassenrath for saying this, but I had to do it the hard way!: in the early days there was no documentation for the CD-QUICKSTATUS command, so I had to do it empirically--start play, do the call, look at the number, and Wow! the track is playing. If you want to be sophisticated, look at the *cdtv.h* file for the QUICKSTATUS bits and write lots of sophisticated code, and have all the debugging fun. If you want to do brute force programming, keep doing QUICKSTATUS calls until you time yourself out OR it returns 101--that's decimal One Hundred And One. This magic number means "I'm here, I'm alive, and I'm playing".

For applications that need frame-by-frame sync to the CD-DA this is a vital step. For "loose" applications that display something, play something, and carry on, it's a waste of effort.

If there is any doubt about the track's validity (see disk popping considerations above) then simply waiting for the Big One O One could leave you with a locked machine.

Synchronizing to the CD-DA

The track is running. Now what? You have a file, say from a music sequencer, in MSF format, starting from an offset of 0:0:0 into the track (improbable, but you get the point)--how do you lock it to the CD-DA?

The best way is to use the CD_FRAMECALL command. This tells the CDTV to handle your code every time a new "frame" of audio data is sent to the audio output. CD frames happen 75 times per second. Unfortunately, like the habit of driving on the "wrong" side of the road, SMPTE timings come in various national flavors. SMPTE frame rates can be (normally) 24, 25 or 30 frames per second. Up to you, or more accurately, up to your parents.

Getting 25 frames a second is easy--your handler code counts in threes and updates a clock in MSF format, remembering that the frames will then run from 1-25, after which you increment seconds. The calculation for 24 or 30 frames is nastier, so I'll leave that to you. The danger point here is that the framecall procedure happens on the supervisor stack, and you had really better not get fancy with it! Just increment a master clock MSF variable or Cause a software interrupt.

The Gotcha factor in the CD_FRAMECALL function is that you SendIO() the StdIO and do nothing else with it until you want to kill it, when you AbortIO() it and *wait until its terminated.*

Having set this up, you are now in a position to check your internal data structures against the master clock which is being incremented by the framecall and do things as and when required.

If you need to use more precise timings then you can use Timer B for microsecond precision (taking PAL/NTSC into account) and maybe adjust a fudge factor by checking the framecall result against expectations periodically.

Generally it's easier to pre-convert standard SMPTE timings into 75f/sec CD frame rates. In fact, as a general rule on the CDTV from my experience, it's better to pre-compute anything you can. You have loads of data space compared to a slow processor running brain-damaged in Chip ram. (For this reason, ACBM format or a variant is much faster way of loading and showing pictures than using compressed ILBMs, or, at least for HAM or EHB photo pictures, the debox routines.) Unpacking a standard Midi file format in real time on the CDTV is not sensible... much better to write a conversion program and massage the data into a quick-load format. A high proportion of your development cycle will be involved in writing tools to maximize data efficiency.

For example: you have a Midi file which has one track running at 60 ppq with 192 Midi clocks per crotchet. To convert the "events" in this file to CD-frame timed format you can see that one crochet lasts exactly one second, and if the first starts at 0 frames the second will start at 75 frames and the third at 150 frames. From which you will conclude that the absolute frame time for any event is a simple conversion of the total elapsed time from the start of the playback. This is fine for tracks without tempo variations, but when there is a tempo change you have to do a bit of juggling with your maths--reset your logical master clock to zero, compute any remaining time on any track event back to Midi clocks, and then redo the sums.

The great advantage of doing this off-line in the sense of before mastering is that you can test it to destruction before committing to the expensive plastic.

PAL-NTSC

The PAL-NTSC conflict remains one of the biggest nightmares on the Amiga/CDTV. Because American developers don't have to worry about it much they don't really understand it, and make dangerous assumptions. One of the great advantages of the CDTV is that the CD frame rate is the same under both PAL and NTSC whereas everything else, VBeams, DMA, processor clock etc. are different. So unless you *really can't*, use the CD frame rate as a clock and not - well - a clock!

An NTSC machine is an NTSC machine. Period. A PAL machine may end up (about 50% of the time with a genlock board in!) thinking it's an NTSC machine when it isn't. The gfxBase flags are *totally and utterly unreliable* about this. The only way to be sure is to clock the BeamPos several thousand times and check if it goes past 299. If it does, whatever the gfxBase flags say, you're running on a PAL machine, and should (a) patch the NormalDisplayRows etc to the PAL standard and (b) use the PAL DMA clock rate divider constants for audio and timer rates. If you don't you will be out of tune, lose the bottom of your screen, and be out of time.

Other Audio Considerations

First, a few common questions:

(Q) Can I just play the left or right CD-DA, hence doubling the mono high-quality playback time to 144 minutes?
(A) No.

(Q) Can I get at the CD-DA 16 bit DACs and play back my own 16 bit internal samples under program control?
(A) No.

(Q) Can I get the CD-DA data into memory?
(A) No. The CD-DA side of the system is sealed for licensing reasons.

(Q) Can I alter the CD-DA volume?
(A) Yes. The CD_MUTE and CD_FADE device calls fully control CD-DA volume level.

(Q) Is the CD-DA audio output level locked to the internal sound generation circuitry?
(A) No. Forget theory about relative DB levels and set your volume ratios when you have a test disk. Theory is expensive in CDTV development. (see DOING IT RIGHT)

(Q) Is there a quick and easy way to make CD+G disks?
(A) If there is, tell me.

(Q) What's the best way to get high quality audio onto the CD?
(A) Coming RSN.

Audio Quality

There are two (or more) issues about audio quality. The first is for CD-DA tracks, the second for internal 8-bit sounds.

Using CD-DA is pointless if you do not produce a professional quality recording. From which it follows that waving a cassette recorder in the air is going to give you some of the most expensive hiss you can reproduce. DAT recorders cost approx 500 in the UK, and, life being what it is, probably about $500 in the USA. DAT recorders work to the same specification as CD-DA and what you hear is what you get.

However, do not make the assumption that either DAT recorders or CD units work at an invariable speed. Speed variations between different drives can be significant--like 2 minutes over 72 minutes, meaning a potential closing velocity of four minutes between two drives! If you rely on stopwatch timings for long files you may run into problems. So the way to cut down problems is: lay the audio tracks, get them edited by the CD manufacturer onto the master tape, make a test pressing and leave them alone.

It is obviously outside the scope of this document to describe audio recording techniques. Suffice it to say that for professional results there is no substitute for a professional recording studio and engineer.

But why use CD-DA? For many purposes it is not necessary. It is expensive, and for simple speech files etc may chew up your disk at an alarming rate. If you want professional quality music there is no alternative. For help files, commentary, the ordinary traffic of multimedia, you may as well use Amiga audio. Bear in mind, though, that you will frequently have to use a background spooling process to play long samples (remember the 128K limit for DMA samples) and this may eat some processor time. The CD-DA is fully asynchronous and leaves you to do whatever you like.

There is a wealth of example programs in the public domain which show you how to play Amiga audio, so I will not go deeply into that. What is rarely mentioned though is the invidious comparison between 16-bit and 8-bit sound that cannot be avoided when you have a disk with CD-DA and 8-bit sounds running consecutively or together. Most Amiga sound demos use Rock music, and sound pretty good. Most CDTV applications use speech, which may not.

Rock music is usually densely-textured and compressed. Audio compression means that the overall signal strength is forced up to something near a constant level. This is very appropriate for the audio hardware, because the noise is effectively buried in the signal. With speech, however, there are very low level signals at the ends of words. Here the amplitude of the waveform drops off until it is hovering around the zero line--in other words, it is revealing the inescapable truth that 8 bits gives you vastly less precision than 16 bits. The low level sounds may crackle or break up around the zero amplitude line as the sampler tries to decide whether this tiny noise level constitutes 1 or 0. Most good sampling software will allow you to ramp the amplitude down a bit when you have made the sample. This may well knock out some rogue bits, but will also introduce new errors at the next level down. You can keep on doing this until you have silence.

The simplest solution to this is to use a gate in the recording studio. This electronic filter will trap any sound in the danger zone around the zero level, and not pass it out of the mixing desk. Gates should be used with caution though, as they can cause a nasty clipping if the threshold is too high. Just a little gate should be enough. It's a matter of personal taste, but I dislike compression on voices, though it's worth adding a little with some speakers. Using a microphone with very high sensitivity to the upper range of frequencies is more likely to accentuate exactly the areas you want to eliminate. What we are doing here is trying to optimize the recording for the hardware on which it will run, not strive for the perfect recording, which is the CD-DA path. You may well have problems with some recording engineers--well, most people have problems with most recording engineers. The engineer is trained to get the highest possible band-width from his studio. You want him to cripple his system so that the final result will sound better on the CDTV than his perfect recording would. If necessary, equalize the top frequencies down a little, avoid sibilance like the plague, and watch out for the wavering zero-line quantization problem.

For a first session or tests it is a good idea to go straight from the desk into an Amiga digitizer. Use the best you can get. Some of the cheaper units work fine at high amplitudes but have poor internal noise rejection--they will be slinging loads of Amiga bus noise into the ADCs.

When working with audio it is important to remember the wide range of equipment performance that will be used by CDTV owners when they play the disk. You could be on a dreadful tv or a megabuck hi-fi system. The acoustic of your room will colour the sound you hear to a very significant degree, so the best method for good quality control is to take the audio output either from the CDTV headphone jack, or on an Amiga via an amp or mixer and use a pair of high quality headphones. They will defeat the room's acoustic, give you good sensitivity to pops crackles and noise. They may be too harsh in fact, but that's better than simply not hearing the problem.

There are other methods of getting digital sound into the system. On Music Maker the instruments were sampled in the studio on the 16-bit Casio S1000 sampler, shaped on (dare I say, an ST) with Avalon, then saved to disk, passed to an Amiga, and computed from 16 to 8 bits with a custom program tool.

There are DAT->disk systems available on some platforms (Mac, PC, ST) and any time now on the Amiga. Their success or failure in this realm of application-building depends entirely on how good they are at converting from 16 to 8 bits, and they do not eliminate the zero line quantization problem unless they are very very smart and have efficient software gates built in.

PAL-NTSC Again

The audio DMA clock rate is different between the two systems. Usually the difference is scarcely audible if you don't have perfect pitch. But if you have CD-DA running a music track and do not check that you are really on a specific model and use the correct the clock divide constant, the track will sound bad.

Local Anaesthesia--or Reducing the Pain

For the first-time CDTV developer the prospect of a read-only test- and-cry medium is daunting. The good news is, it stays daunting. The early history of CDTV is littered with very expensive and totally useless CDs. I know... I have twenty in a box right next to me!

The only way to keep the terror down to manageable proportions is a sound methodology. If you are going to make a mixed-mode disk you will need an audio proof disk at the earliest possible stage. Trying to cut this stage out of the loop will cost more, not less. Get the audio finalized, cleared for world copyright, recorded and then get it onto a CD. If at the same time you can get all the data onto the CD, do it, because it makes life easier. If the audio is on and the data are on then you can hone your program(s) with nothing more than a floppy drive plugged into the CDTV. This is the preferred method, though if you if you cannot build the whole thing at this stage you may have to use a SCSI board in the CDTV expansion port and a SCSI device hung onto that.

As mentioned before, when you lay your audio, allow for the data track, even if it means that track one lasts fifty minutes and contains nothing but silence.

Use a good CD manufacturer. The good companies know everything and more about the physics of the disk, the ISO9660 standard, audio balancing, the works. The bad ones have a Yamaha One-Shot machine and a lots of potted plants.

As with all other CDTV disks, the worst element after testing, software finalization etc, is making sure that the *startup-sequence* and prefs settings are correct. This is very important after working from SCSI or floppy boot, because you will be using the *startup-sequence* on the floppy or SCSI, which may have different assigns on it.

During development you may want to address files on the floppy or the SCSI rather than the CD. My preferred method is use paths to logical devices in the code. For example CD99: Your startup on floppy of SCSI will assign that to itself most probably... but in the final system you simply have assign CD99: CD0: in the *startup-sequence*. This really does make life easier.

Last Thoughts

The *cdtv.device* is powerful. It does nearly all the work for you. The biggest rule and the most important is make sure your file control is 100% and make sure the audio is stopped and cleaned up before continuing, remembering that interrupt calls from VBeam or FrameCall may need some time to disable. It's also a good idea to make sure that you can never get into the situation where you try to access a picture file or something like that while the audio is running. This is a guaranteed way to lock up the system.

With the above things in mind, the system becomes easy and nice to use. It's a real pleasure to hear a high-quality CD-DA track come up and play under your program control, and all you need is a few lines of code.

You can end up swapping CD disks often during development. If you start to get increasing errors it almost certainly means your CD is dirty. Take into the bathroom, wash with soapy water, dry with a towel, and shove it back in.

Try doing that with a floppy!

◆

THE ZORRO III BUS SPECIFICATION

A General Purpose Expansion Bus for
High Performance Amiga Computers

Document Revision 1.10

Vernal Equinox Release

by Dave Haynie
March 20, 1991

Copyright © 1990, 1991 Commodore-Amiga, Inc.

IMPORTANT INFORMATION

"A life spent making mistakes is not only more honorable but more useful than a life spent doing nothing."
-George Bernard Shaw

This Document Contains Preliminary Information

The information contained here, while a honest attempt to get as much Zorro III information down on paper as early and accurately as possible, is still somewhat preliminary in nature and subject to possible errors and omissions. Being early in the life of the Zorro III bus, very few Zorro III cards have yet been designed, so some features described here have not actually been tested in a system, or in some cases, actually implemented as of this writing. That, of course, is one major reason for having a specification in the first place.

Commodore Technology reserves the right to correct any mistake, error, omission, or viscious lie. Corrections will be published as updates to this document, which will be released as necessary in as developer-friendly a manner as possible. Revisions will be tracked via the revison number that appears on the front cover. New revisions will always list the corrections up front, and developers will be kept up to date on released revisions via the normal CATS channels.

All information herein is Copyright © 1990, 1991 by Commodore-Amiga, Inc., and may not be reproduced in any form without permission.

ACKNOWLEDGEMENTS

> *"Art is I; science is we."*
> *-Claude Bernard*

I'd like to acknowledge the following people and groups, without whom this new stuff would have been impossible:

- The original Amiga designers, for designing the first microcomputer bus with support for multiple masters, software board configuration, and room to grow.
- The rest of the A3000 Engineers: Greg Berlin, Hedley Davis, Scott Hood, and Scott Schaeffer; PCB master Terry Fisher; and the lab maniacs George Terbush, Brian Fenimore, and Dan Faust. And of course the A3000 boss men, Jeff Porter and Henri Ruben, who let it all happen.
- The folks who helped review the original version of this document, overnight: Joe Augenbraun, Dan Baker, Hedley Davis, Bryce Nesbitt, and Jeff Porter. And to the numerous folks who've helped out with questions, corrections, and other feedback ever since.
- The Commodore-Amiga software group, and the Commodore Semiconductor Group, for excellent support in their respective areas. Though, about that Rev H chip.....
- Commodore's Developer Support people from both sides of the Atlantic.
- Gold Disk, for some good and relatively bug free electronic publishing software.
- Iggy; an excellent cat, an excellent foot warmer.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 Intended Audience........ 1-1
1.2 Bug Reports........ 1-2
1.3 Amiga Bus History........ 1-2
1.4 The Zorro III Rationale........ 1-3
1.5 Document Revision History........ 1-4
1.5.1 Changes for Rev 0.90........ 1-5
1.5.2 Changes for Rev 0.91........ 1-5
1.5.3 Changes for Rev 1.00........ 1-5
1.5.4 Changes for Rev 1.01........ 1-5
1.5.5 Changes for Rev 1.10........ 1-5

CHAPTER 2 ZORRO II COMPATIBILITY

2.1 Changes From The A2000 Bus........ 2-2
2.1.1 6800 Bus Interface........ 2-2
2.1.2 Bus Memory Mapping and Cache Support........ 2-2
2.1.3 Bus Synchronization Delays........ 2-3
2.1.4 Zorro II Master Access to Local Slaves........ 2-3
2.1.5 Bus Arbitration and Fairness........ 2-3
2.1.6 Intelligent Cycle Spacing........ 2-3
2.1.7 Bus Drive and Termination........ 2-4
2.1.8 DMA Latency and Overlap........ 2-4
2.1.9 Power Supply Differences........ 2-4
2.2 Bus Architecture........ 2-5
2.3 Signal Description........ 2-5

2.3.1 Power Connections........ 2-6
2.3.2 Clock Signals........ 2-6
2.3.3 System Control Signals........ 2-7
2.3.4 Slot Control Signals........ 2-9
2.3.5 DMA Control Signals........ 2-9
2.3.6 Addressing and Control Signals........ 2-11

CHAPTER 3 BUS ARCHITECTURE

3.1 Basic Zorro III Bus Cycles........ 3-1
3.1.1 Design Goals........ 3-2
3.1.2 Simple Bus Cycle Operation........ 3-2
3.2 Advanced Mode Support Logic........ 3-4
3.2.1 Bus Locking........ 3-4
3.2.2 Cache Support........ 3-5
3.3 Multiple Transfer Cycles........ 3-5
3.4 Quick Bus Arbitration........ 3-7
3.5 Quick Interrupts........ 3-9
3.6 Compatibility with Zorro II Devices........ 3-10

CHAPTER 4 SIGNAL DESCRIPTION

4.1 Power Connections........ 4-1
4.2 Clock Signals........ 4-2
4.3 System Control Signals........ 4-2
4.4 Slot Control Signals........ 4-4
4.5 DMA Control Signals........ 4-5
4.6 Address and Related Control Signals........ 4-5
4.7 Data and Related Control Signals........ 4-7

CHAPTER 5 TIMING

5.1 Standard Read Cycle Timing........ 5-2
5.2 Standard Write Cycle Timing........ 5-4
5.3 Multiple Transfer Cycle Timing........ 5-6
5.4 Quick Interrupt Cycle Timing........ 5-8

CHAPTER 6 ELECTRICAL SPECIFICATIONS

6.1 Expansion Bus Loading........ 6-1
6.1.1 Standard Signals........ 6-2
6.1.2 Clock Signals........ 6-2
6.1.3 Open Collector Signals........ 6-3
6.1.4 Non-bussed Signals........ 6-3
6.2 Slot Power Availability........ 6-3
6.3 Temperature Range........ 6-3

CHAPTER 7 MECHANICAL SPECIFICATIONS

7.1 Basic Zorro III PIC........ 7-2
7.2 PIC with ISA Option........ 7-3
7.3 PIC with Video Option........ 7-4

CHAPTER 8 AUTOCONFIG®

8.1 The AUTOCONFIG® Mechanism........ 8-1
8.2 Register Bit Assignments........ 8-2

APPENDICES

A.1 Physical and Logical Signal Names........ A-1
A.2 A Glossary of Terms........ A-5
A.3 Zorro III Implementations........ A-9

TABLES AND FIGURES

Figure 1-1 A3000 Memory Map 1-4

Figure 2-1 A2000 vs. A3000 Bus Termination 2-5
Figure 2-2 Expansion Bus Clocks 2-7
Figure 2-3 Zorro II Bus Arbitration 2-10

Figure 3-1 Basic Zorro III Cycles 3-3
Figure 3-2 Multiple Transfer Read Cycles 3-5
Figure 3-3 Zorro III Bus Arbitration 3-8
Figure 3-4 Interrupt Vector Cycle 3-9
Figure 3-5 Zorro II Within Zorro III 3-10

Table 4-1 Memory Space Type Codes 4-5

Table 6-1 Zorro III Drive Types 6-2

Figure 8-1 Configuration Register Mapping 8-2

CHAPTER 1

INTRODUCTION

"Welcome, my son. Welcome to The Machine."
-Pink Floyd

This document describes the complete Zorro III bus, first implemented in the Amiga 3000 Computer. The Zorro III bus is a performance 32 bit expansion bus that is also upward compatible with the Zorro II bus (Amiga 2000 expansion bus). The main intent of the Zorro III bus is to allow fast 32 bit peripherals and memory devices to be added to a high performance Amiga, such as the Amiga 3000, while at the same time allowing standard Zorro II devices to be used wherever they make sense in such a system. This compatibility also insures that the Amiga 3000 will have a number of hardware and software compatible expansion devices available upon introduction, and that Amiga 2000 owners will be able to take their expansion card investment along with them should they migrate to a higher performance Amiga.

1.1 Intended Audience

This document was written primarily for hardware engineers interested in designing Plug In Cards for the Zorro III expansion bus. While it may occasionally be of use to software engineers interfacing to such Zorro III PICs, Amiga system software provides an interface layer (*expansion.library* in the Amiga OS) which manages the needs of most card-level software. A reasonable level of microcomputer knowledge is prerequisite to get much meaning out of these pages. A good understanding of the Motorola 680x0 processors will be quite useful, as will be an understanding of the Zorro II expansion bus used on earlier Amiga computers such as the Amiga 2000.

1.2 Bug Reports

This is the second major publication of the *Zorro III Bus Specification.* While every effort has been made to keep it as accurate as possible, there is certainly the possibility that some errors have made it into this document. Anyone finding any error is encouraged to contact Commodore at the address below:

Dave Haynie/A3000 Systems Engineering
1200 Wilson Drive
West Chester, PA 19380

Bugs can also be reported on BIX or via Usenet; on BIX, use the "amiga.com/hardware" conference, or contact Dave Haynie directly as "hazy"; for Usenet users, bug reports can be sent to the address "{uunet,rutgers}!cbmvax!bugs" (use "cbmvax.cbm.commodore.com" if you like domain names); please also copy any such reports to "{uunet,rutgers}!cbmvax!daveh".

1.3 Amiga Bus History

The original Amiga computer, the Amiga 1000, was introduced in 1985. While it had no built-in standard for expandability, the capability for some form of expansion was considered extremely important; personal computer history up to that date had shown several times that an open hardware expansion capability was often critical to a personal computer's success and to its capability to adapt to new or unusual applications. The A1000 was designed with a connector giving access to the internal 68000 bus and a few other system signals. Shortly after introduction, the formal expansion specification for a card chassis that would connect to the A1000 was published. This bus became commonly known as the Zorro bus*. While the backplane specification was very easy to implement with 1985 PAL technology based on the existing 68000 signals, the specification did incorporate a number of advanced features. Far more sophisticated than the IBM-XT/AT and Apple II buses in common use at the time, the Zorro bus allowed any slot to master the bus, and it linked expansion cards with the system software. Addressing jumpers were eliminated, the card's address instead being assigned by software, and cards could easily be identified by software and linked with appropriate driver programs, all with a minimum of user intervention.

With the introduction of the Amiga 2000 system, the Zorro bus was changed slightly. Additional discrete interrupt lines were added, replacing the encoded lines that couldn't easily be used by any bus resident device. As it turns out, these additional encoded lines weren't any more useful, as they couldn't be disabled by software, and as such, they're no longer considered an official part of the Zorro II bus specification (they are supported as part of Zorro III). Finally, the form factor was changed to match that of the IBM PC-AT card, acting as both a cost reduction and allowing the Zorro II bus to offer the PC-AT bus as one optional secondary bus extension. This modified specification became commonly known as the Zorro II bus, and it's the

---

* The original "Zorro" name comes from the code name of one of the A1000 prototype boards. The "Zorro" board was the one that followed the "Lorraine", and was the board in the works when much of the expansion specifications were worked up. Since everyone uses the "Zorro" name, and no one's suggested a better name, I stick with it throughout this document.

Amiga bus standard that's been in use for most of the Amiga's life. And it's a bus standard that will continue to be important.

1.4 The Zorro III Rationale

With the creation of the Amiga 3000, it became clear that the Zorro II bus would not be adequate to support all of that system's needs. The Zorro II bus would continue to be quite useful, as the current Amiga expansion standard, and so it would have to be supported. A few unused pins on the Zorro II bus and the option of a bus controller custom LSI, gave rise to the Zorro III design, which supports the following features:

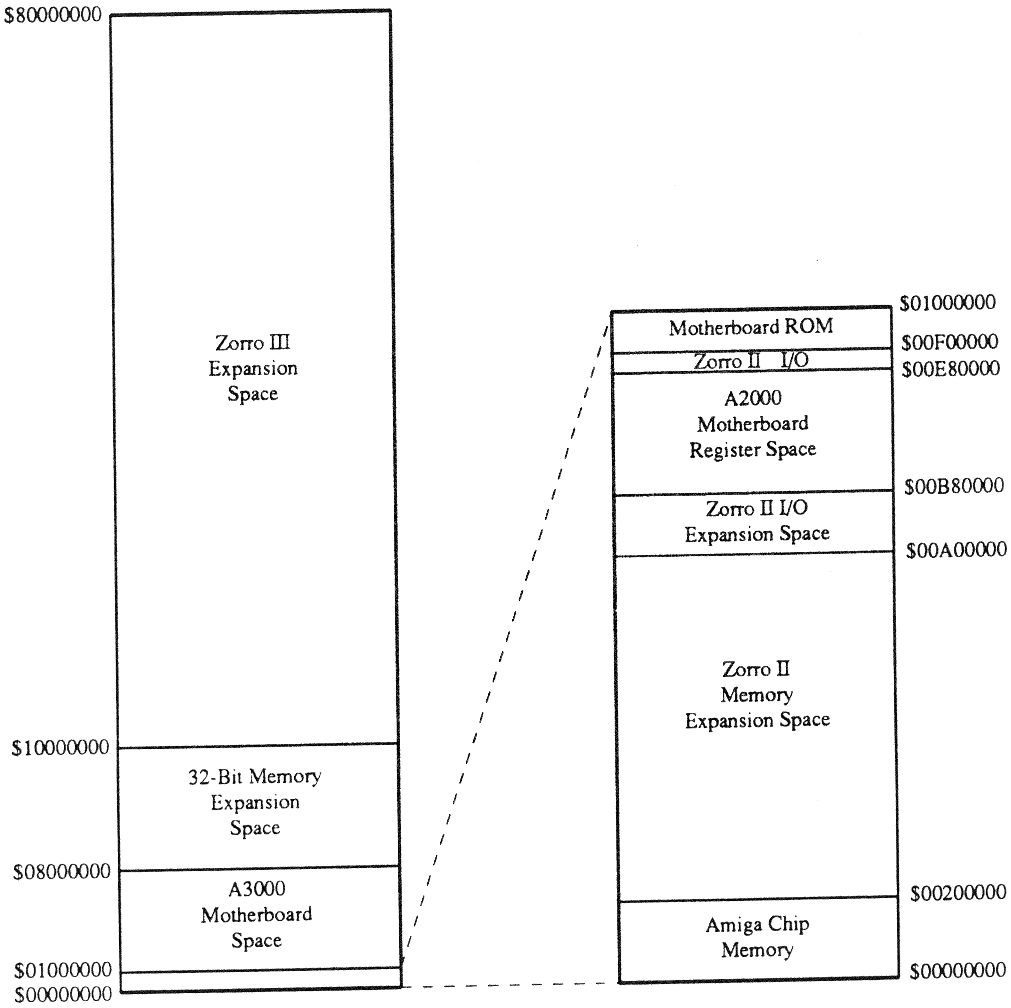
- Compatibility with all Zorro II devices.
- Full 32 bit address path for new devices.
- Full 32 bit data path for new devices.
- Bus speed independent of host system CPU speed.
- High speed bus block transfer mode.
- Bus locking for multiprocessor support.
- Cache disable for simple cache support.
- Fair arbitration for all bus masters.
- Cycle by cycle bus abitration mode.
- High speed interrupt mode.

Some of the advanced features, such as burst modes, are designed in such a way as to make them optional; both master and slave arbitrate for them. In addition, it is possible with a bit of extra cleverness, to design a card that automatically configures itself for either Zorro II or Zorro III operation, depending on the status of a sensing pin on the bus.

The Zorro III bus is physically based on the same 100 pin single piece connector as the Zorro II bus. While some bus signals remain unchanged throughout bus operation, other signals change based on the specific bus mode in effect at any time. The bus is geographically mapped into three main sections, *Zorro II Memory Space, Zorro II I/O Space,* and *Zorro III Space.* The memory map in *Figure 1-1* shows how these three spaces are mapped in the A3000 system. The Zorro II space is limited to a 16 megabyte region, and since it has DMA access by convention to chip memory, it is in the original 68000 memory map for any bus implementation. The Zorro III space can physically be anywhere in 32 bit memory.

The Zorro III bus functions in one of two different major modes, depending on the memory address on the bus. All bus cycles start with a 32 bit address, since the full 32 bit address is required for proper cycle typing. If the address is determined to be in Zorro II space, a Zorro II compatible cycle is initiated, and all responding slave devices are expected to be Zorro II compatible 16 bit PICs. Should a Zorro III address be detected, the cycle completes when a Zorro III slave responds or the bus times out, as driven by the motherboard logic. It is very important that no Zorro III device respond in Zorro III mode to a Zorro II bus access; as the following chapters will reveal, the two types of cycles make very different use of many of the expansion bus lines, and serious buffer contention can result if the cycle types are somehow mixed up. The Zorro III bus of course started with the Zorro II bus as its necessary base, but the Zorro III bus mechanisms were designed as much as possible to solve specific needs for high end Amiga systems, rather than extend any particular Zorro II philosophy when that philosophy no longer made any sense. There are actually several variations of the basic Zorro III cycle, though they all work on the same principles. The variations are for optimization of cycle times and for service of interrupt vectors. But all of this in due time.

Figure 1-1: A3000 Memory Map


$80000000
Zorro III Expansion Space
$10000000
32-Bit Memory Expansion Space
$08000000
A3000 Motherboard Space
$01000000
$00000000
$01000000
Motherboard ROM
$00F00000
Zorro II I/O
$00E80000
A2000 Motherboard Register Space
$00B80000
Zorro II I/O Expansion Space
$00A00000
Zorro II Memory Expansion Space
$00200000
Amiga Chip Memory
$00000000

1.5 Document Revision History

While there's significantly more real Zorro III hardware actually in existence at the time of this writing than when the first revision of this document was created, various Zorro III issues are still, from time to time, changing. In order to document these changes, this section was created. Although revision histories often discuss revisions in reverse chronological order, it's done here in chronological order to keep the subsection numbers consistent between revisions of this document.

1.5.1 Changes for Rev 0.90

The major changes in Rev 0.90 are actually additions. Specifically, the remaining parts of the Zorro III Timing (Chapter 5) and Mechanical (Chapter 7) specifications have been incorporated into this document. Additionally, the Zorro III design example in Appendix A.4 has been deleted. This simple and somewhat kludgy example has been surplanted by a more useful, straightforward, and throughly explained example, available as the separate document *BIGRAM 8/32: A Complete Zorro III Design Example*. In general, we expect both documents to be distributed together, but as always, CATS can assist in the procurement of any missing information.

1.5.2 Changes for Rev 0.91

In the Introduction (Chapter 1), the official revision history has been added as a standard part of this document. The Zorro III Bus Architecture (Chaper 3), section 3.5, has been changed to reflect the revised Quick Interrupt vector allocation mehanism. In the Timing specification (Chapter 5), corrections have been made: timing parameter 6 was left out of the section 5.3 timing, and timing parameter 19 was incorrectly specificed in section 5.4. In the AUTOCONFIG® specification (Chapter 8), corrections have been made to the addressing tables for registers **44** and **48**. Also the Quick Interrupt Enable bit (register **08**:4) and Vector Register (register **50**) have been deleted from the specification. Quick Interrupt Vector allocation is now handled via an Exec call, a single configuration unit can have several vectors, and the means of storage on a PIC is up to the designer.

1.5.3 Changes for Rev 1.00

In the AUTOCONFIG® specification (Chapter 8), bit 4 of register **08** has been changed to always read 1 for Zorro III PICs. This change was necessary for compatibility with 1.3, due to a bug in the 1.3 expansion.library. Also, the nybble write configuration mode for the Zorro III configuration block has been eliminated, only byte and word writes are now supported.

1.5.4 Changes for Rev 1.01

The AUTOCONFIG® specification change listed in 1.5.3 was missing from Chapter 8 in Rev 1.00 of the spec, now it's actually there. Additionally, some clarification on the proper action of slave cards and bus error conditions has been added to Chapter 4.

1.55 Changes for 1.10

The /INT$_1$, /INT$_4$, /INT$_5$, and /INT$_7$ lines have been eliminated from the Zorro III bus specification. Although the A3000 hardware supports these, some AmigaOS software conventions makes their use impossible under the AmigaOS. These lines are now considered reserved. Also, in section 3.5, the vector poll command code was given as 16, where it's actually 15; this has been corrected.

CHAPTER 2

ZORRO II COMPATIBILITY

> *"In Jersey anything's legal, as long as ya don't get caught."*
> *- Traveling Wilburys*

The A3000 bus is a rather extensive superset of the A2000 bus design. The compatibility is based on distinct bus modes, rather than a simple extension to the existing bus mechanisms. Through the use of an integrated bus controller (the Fat Buster chip), the expansion bus configures itself differently for the 16 bit A2000-compatible Zorro II modes than the 32 bit Zorro III modes. As a result, while there are still only 100 pins on the expansion bus, some pins change function considerably depending on the bus activity that's currently in progress. While the Zorro II modes of the Zorro III bus are as compatible as possible with the Zorro II bus specification (especially the A2000 implementation of this specification), there are some small differences between the two expansion buses.

Aside from these differences, in general, it's important to understand the Zorro II bus in order to understand the Zorro III bus. The general features of the A3000 bus, like autoconfiguration, the master-slave bus architecture, and the physical attributes come from the Zorro II expansion bus. Other features of the Zorro III bus address shortcomings of the Zorro II architecture, but Zorro II has a hand in how some of these shortcomings are solved under Zorro III. Those with a full understanding of the Zorro II bus will mainly be concerned with the possible bus incompatibilities listed here.

2.1 Changes From The A2000 Bus

While much effort has been made to assure that the Zorro II mode of the A3000 bus is as compatible as possible with the A2000 bus, there are a few points to consider here. Primarily, the A3000's Zorro II modes are driven with a state machine that emulates the 68000 bus protocol. This emulation must be based on the published Motorola specifications detailing 68000 bus behavior. While this has the interesting effect of changing the Zorro II bus from CPU dependent to CPU independent, there's some margin for trouble. Zorro II PICs also designed to these specifications should have no trouble in the A3000 bus in most cases. However, anything designed based on observed 68000 behavior rather than documented 68000 operation is at serious risk of failing in an A3000 bus, as one might expect. There are also actual documented differences, which are listed below.

2.1.1 6800 Bus Interface

A major difference between the A3000 expansion bus in Zorro II mode and the A2000 bus are the absence of the signals /VPA and /VMA, which comprise the 6800/6502 peripheral support mechanism that's part of the 68000 bus interface. This mechanism was never a supported part of the Zorro II specification, however, and it should not be used by any PIC. Any Zorro II PIC that depends on /VPA or /VMA will not work in the A3000 bus. It was, in fact, impossible to legally use this on the A2000 bus. The E clock is, however, supported on the Zorro III bus, though its duty cycle may vary in some situations.

2.1.2 Bus Memory Mapping and Cache Support

Another change to the Zorro II implementation is that the bus mapping logic works a little differently. Zorro II address space is broken up into memory and I/O address space. Memory space is the standard 8 megabyte space from $00200000-$009FFFFF. The I/O address space is mapped at $00E80000-$00EFFFFF, and a new 1.5 megabyte section (previously reserved for motherboard devices) from $00A00000-$00B7FFFF. Zorro II cycles are not generated for non-Zorro II address space, even for 68000 space resources on the local bus. So, for example, a CPU access to chip memory would be visible to a Zorro II PIC in an A2000 backplane, but invisible to that same PIC in an A3000 backplane. Since this extra information on the Zorro II backplane can't be legally used by any PIC anyway, it should not be used by any existing A2000 PICs.

The reason for the two distinct mapping regions is for cache support of Zorro II PICs. All access by the local bus* master to Zorro II memory space results in the local bus cache enable signal being driven and a full port read (eg, both bytes) regardless of the actual data transfer size being requested. A local bus access to Zorro II I/O space results in the local bus cache disable signal being driven and the data strobes for reads indicating the requested transfer size. This cache mapping mechanism was first implemented in the A2630 coprocessor card, so it's not an entirely new concept.

---

*The *local bus*, motherboard bus, and CPU bus are the same thing; the immediate 680x0 bus connected directly to the CPU in an Amiga computer. Current Amiga computers typically support three distinct buses; the expansion bus, local bus, and chip bus. From the point of view of the expansion bus, the local and chip buses appear as a unified device which may be master or slave to the expansion bus.

2.1.3 Bus Synchronization Delays

Due to the asynchronous nature of the local-to-expansion bus interface for Zorro II cycles, extra wait states may occasionally be added for local to expansion or expansion to local cycles. These are generally manifested as delays between consecutive cycles, since the bus controller is not going to require extra waiting during the cycle -- things will have already been synchronized at that point. The synchronization problems get more difficult for Zorro II master access to local bus slaves, and as a result, wait states here are very common. The actual number of wait states generated in any case will be based on the particular implementation.

2.1.4 Zorro II Master Access to Local Slaves

The only supported local bus resource that's guaranteed accessible to a Zorro II expansion bus master as a slave device is chip bus memory. All I/O devices are implementation dependent and not supportable via DMA. Any attempted access to unsupported local bus resources as expansion slaves will result in an error condition being signalled on both the local and the expansion buses. Most other local bus resources, such as local bus fast memory, are located outside of Zorro II space on most systems and obviously not available to Zorro II masters.

2.1.5 Bus Arbitration and Fairness

The Zorro II bus is now arbitrated fairly. The normal slot-based order of precedence is given to requesting devices, just as in the A2000 implementation. As always, once a bus master assumes bus mastership, it has the bus for as long as it wants the bus (of course, trouble can result if a device takes the bus over for too long). Once a master gives up the bus, it will not be granted it back until all subsequent requests have been serviced. Bus arbitration at its best will be slightly slower than in the A2000 implementation, due to the fairness logic, but it is impossible to jam the arbiter with asynchronous bus requests as in the A2000. The new style arbiter also holds off bus grants while hidden local bus cycles are in progress, so there's no guarantee of a minimum time between bus request and bus grant specified.

2.1.6 Intelligent Cycle Spacing

In order to permit a free intermix of Zorro II and Zorro III cycles, the bus control logic is capable of making intelligent decisions when spacing bus cycles. In somc cases, a Zorro II cycle has some component that would naturally extend into a following cycle. The cycle spacing logic detects such a condition, and refuses to start a new cycle until the current one is complete, even if this extends beyond the defined bounds of a Zorro II cycle. For Zorro II PICs that really follow the Zorro II specifications, this should have no effect. However, any Zorro II PIC that holds signals much beyond the end of a cycle, especially critical signals like /SLAVE and /DTACK, will likely incur additional wait states on the Zorro III bus. This is not intended as a license for making sloppy expansion card designs, just an acknowledgement that some Zorro II devices may cause a conflict with the faster Zorro III bus timings, and the best thing to do about such cases is to make them work, even with a possible performance penalty.

2.1.7 Bus Drive and Termination

Finally, the Zorro III bus uses different bus termination than that in the A2000. The Zorro II specification didn't specify the termination expected; backplanes were built that didn't even have termination. The A2000 bus used a circuit consisting of a capacitor in series with a resistor to ground for most of the bus signals. This has good reflection cancelling properties without increasing crosstalk (a major concern on the 2-layer A2000 motherboard), but it does slow things down measureably. The main reason for the change on the A3000 backplane is to support the faster Zorro III bus modes. The multi-layer A3000 motherboard permits a reasonably high current bus without undue crosstalk. The thevenin termination makes switching logic levels start from a midpoint instead of a rail, especially for a bus coming out of tri-state (which, based on the Zorro III design, happens constantly). This should not cause problems with Zorro II cards, but it's conceivable that some cards may need to be adjusted to work in this bus (the Zorro III bus requires somewhat higher current capability than the Zorro II bus does. The A3000 does not support enough slots for loading to be a likely problem, but future Zorro III backplanes will have more slots and make this an important consideration).

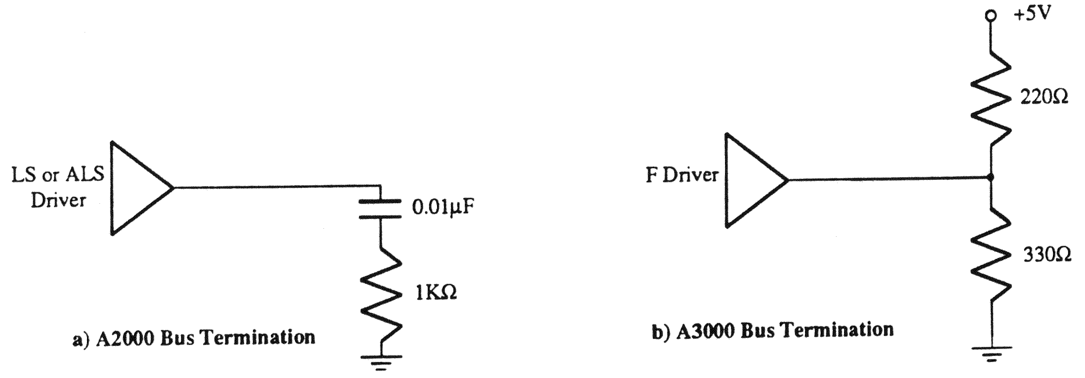


Figure 2-1: A2000 vs. A3000 Bus Termination

2.1.8 DMA Latency and Overlap

Zorro II bus masters in a Zorro III backplane will, in many cases, receive a bus grant much sooner than they would in a standard Zorro II backplane. Additionally, in some cases, expansion bus cycles will overlap local bus cycles. The latency incurred on the Zorro II bus during heavy custom chip activity has been greatly reduced for any Zorro III bus master. This should be transparent to the card in question, though it's a good thing to be aware of.

2.1.9 Power Supply Differences

The Zorro II bus is defined as supplying +5VDC @ 2 Amps to each slot, with one slot per backplane supplying 5.0VDC @ 4.0 Amps. The Zorro III bus only provides the 5.0VDC @ 2.0 Amps for each slot.

2.2 Bus Architecture

The Zorro II bus is a simple extention of the 68000 processor bus. Those without a good knowledge of the 68000 local bus will find *The 68000 User's Manual* from Motorola an excellent reference for many Zorro II issues. The *A500/A2000 Technical Reference Manual* from Commodore-Amiga is also required reading for any Zorro II design issues, as it includes a complete description of all the Commodore-Amiga details that aren't part of the 68000 specification.

The basic Zorro II bus is a buffered version of the 68000 processor bus, physically provided on a 100 pin one-piece connector. The bus is 16 bits wide, and provides 24 bits of addressing information. A bus cycle looks exactly like a 68000 bus cycle. The cycle is defined by an address strobe, terminated by a data transfer strobe, and qualified by a read/write strobe, some memory space qualifiers, and one or two byte selection strobes. The basic bus cycle runs for a total of four cycles of a 7.16MHz clock, though it can be extended to add wait states when required.

The Zorro II bus adds a number of features to the basic 68000 CPU bus. It supplies some Amiga system signals that are useful for expansion card designs, such as many of the Amiga system clocks. The bus provides a default data transfer signal, which expansion cards can easily use and modify rather than go to the trouble of creating their own. It provides a number of discrete interrupt lines which are mixed to provide the 68000 with its standard encoded interrupts. The 68000 bus arbitration protocol is used to allow multiple bus masters; arbitration of the bus requests are managed by the Zorro II bus controller to avoid contention between multiple masters. And of course the bus supplies a number of supply voltages for powering cards.

A powerful aspect of the Zorro II bus is its convention for automatically configuring expansion cards, AUTOCONFIG®. On system powerup, the system software interrogates each board to determine what kind of board is installed and how much memory space it needs on the bus. The software then tells each board where to reside in memory. The bus provides hardware lines to allow the boards to be configured in a daisy chained fashion regardless of which slots they occupy and to prevent damage to boards if accidently configured to reside at the same memory location. Firmware standards also permit software to autoboot or autoinitialize any board, to match soft-loaded device drivers with individual boards, and to link memory boards into the appropriate system memory lists.

2.3 Signal Description

The Zorro II bus can be broken down into various logical signal groups. Some of these groups are unchanged in the Zorro III bus modes, others are drastically different. This section makes note of the original Zorro II name for each signal and the current Zorro III physical pin name for each signal, where different. Some of this information will be repeated in the Zorro III chapters, where appropriate; nothing in this chapter is considered critical to understanding the Zorro III bus, but it is useful. As previously mentioned, the A2000 bus signals unsupported by the Zorro II specification have been deleted from the Zorro III specification and the A3000 implementation of Zorro III; this section will, however, document those signals for reference purposes. Please see Appendix A for a complete list with pin numbers of the various logical signals that appear on the physical bus during the different phases of the Zorro II and Zorro III bus cycles.

2.3.1 Power Connections

The Zorro III expansion bus provides several different voltages designed to supply expansion devices. There are no changes here that affect Zorro II cards.

Digital Ground (Ground)

This is the digital supply ground used by all expansion cards as the return path for all expansion supplies.

Main Supply (+5VDC)

This is the main power supply for all expansion cards, and it is capable of sourcing large currents; each expansion slot can draw up to 2.0 Amps @ +5VDC. The extra power for one card in any backplane drawing up to 4.0 Amps @ +5VDC is no longer supported.

Negative Supply (-5VDC)

This is a negative version of the main supply, for small current loads only. There is no maximum load specified for the Zorro II bus on a per-slot basis; the A2000 implementation specifies 0.3 Amps @ -5VDC for the entire system.

High Voltage Supply (+12VDC)

This is a higher voltage supply, useful for communications cards and other devices requiring greater than digital voltage levels. This is intended for relatively small current loads only. There is no maximum load specified for the Zorro II bus on a per-slot basis; the A2000 implementation specifies 8.0 Amps @ +12VDC for the entire system, most of which is normally devoted to floppy and hard disk drive motors, not slots.

Negative High Supply (-12VDC)

Negative version of the high voltage supply, also commonly used in communications applications, and similarly intended for small loads only. There is no maximum load specified for the Zorro II bus on a per-slot basis; the A2000 implementation specifies 0.3 Amps @ -12VDC for the entire.

2.3.2 Clock Signals

The Zorro III expansion bus provides clock signals for expansion boards. These clocks are for synchronous Zorro II designs and for other synchronous activity such as bus arbitration. While originally based on Amiga local bus clocks, these have no guaranteed relationship to any local bus activity in newer Amiga computers, but are maintained in Amiga computers as part of the expansion bus specifiation. The relationship between these clocks is illustrated in *Figure 2-2*.

/C1 Clock

This is a 3.58 MHz clock (3.55 MHz on PAL systems) that's synched to the falling edge of the 7M system clock.

/C3 Clock

This is a 3.58 MHz clock (3.55 MHz on PAL systems) that's synched to the rising edge of the 7M system clock.

CDAC Clock

This is a 7.16 MHz system clock (7.09 MHz on PAL systems) which trails the 7M clock by 90° (approximately 35ns).

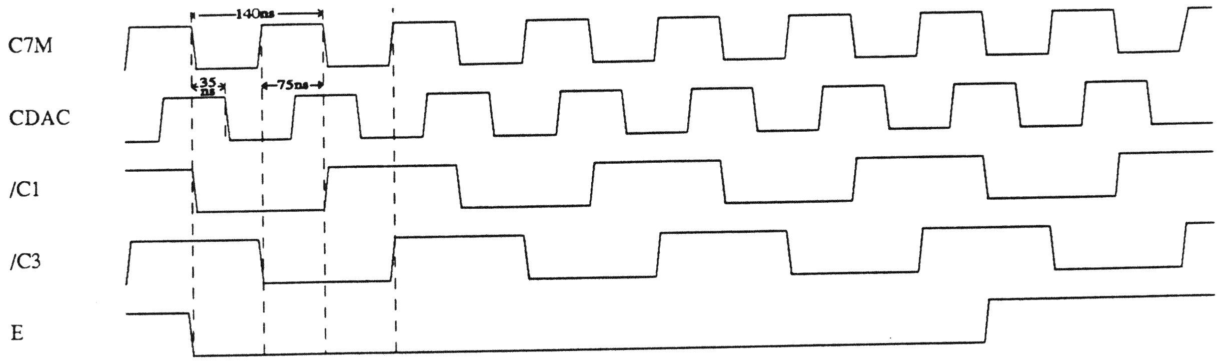


Figure 2-2: Expansion Bus Clocks

E Clock

This is the 68000 generated "E" clock, used for 6800 family peripherals driven by "E" and 6502 peripherals driven by Φ2. This clock is four 7M clocks high, six clocks low, as per the 68000 spec. Note that the bus does not support the rest of the 68000's 6800/6502 compatible interface; there may be better ways to clock such devices.

7M Clock

This is the 7.16 MHz system clock (7.09 MHz on PAL systems). This clock forms the basis for all Zorro II/68000 compatible activity, and for various other system functions, such as bus arbitration.

2.3.3 System Control Signals

The signals in this group are available for various types of system control; most of these have an immediate or near immediate effect on expansion cards and/or the system CPU itself.

Bus Error (/BERR)

This is a general indicator of a bus fault condition. Any expansion card capable of detecting a hardware error relating directly to that card can assert /BERR when that bus error condition is detected, especially any sort of harmful hardware error condition. This signal is the strongest possible indicator of a bad situation, as it causes all PICs to get off the bus, and will usually generate a level 2 exception on the host CPU. For any condition that can be handled in software and doesn't pose an immediate threat to hardware, notification via a standard processor interrupt is the better choice. The bus controller will drive /BERR in the event of a detected bus collision or DMA error (an attempt by a bus master to access local bus resources it doesn't have valid access permission for). All cards must monitor /BERR and be prepared to tri-state all of their on-bus output buffers whenever this signal is asserted. The current bus master should, if possible, retry the bus cycle after /BERR is negated unless conditions warrant otherwise. Since any number of devices may assert /BERR, and all bus cards must monitor it, any device that drives /BERR must drive with an open collector or similar device capable of sinking at least 12ma, and any device that monitors /BERR should place a minimal load on it (1 "F" type load or less). This signal is pulled high by a passive backplane resistor.

System Reset (/RST, /BUSRST) ≡ (/RESET, /IORST) for Zorro III

The bus supplies two versions of the system reset signal. The /RST signal is bidirectional and unbuffered, allowing an expansion card to hard reset the system. It should only be used by boards that need this reset capability, and is driven only by an open collector or similar device. The /BUSRST signal is a buffered output-only version of the reset signal that should be used as the normal reset input to boards not concerned with resetting the system on their own. All expansion devices are required to reset their autoconfiguration logic when /BUSRST is asserted. This signal is pulled high by a passive backplane resistor.

System Halt (/HLT)

This signal is similar to the 68000 processor halt signal, and is driven by a PIC with an open-collector or similar gate only. Its main use is to indicate a full-system reset. Based on the 68000 conventions, an I/O-only reset, such as initiated by the 680x0 RESET instruction, will drive only /RST and /BUSRST on the bus. A full-system reset, such as a powerup reset or a keyboard reset, drives /HLT low as well. PICs that wish to reset the system CPU as well as the bus and I/O devices drive /RST and /HLT, some bus devices such as processor cards may internally reset only on full-system resets. This signal is pulled high by a passive backplane resistor.

System Interrupts

Six of the decoded, level sensitive 680x0 interrupt inputs were originally available on the expansion bus, and these are labelled as /INT$_2$, /INT$_6$, /EINT$_1$, /EINT$_4$, /EINT$_5$, /EINT$_7$ on the Zorro II bus. Only the /INT$_2$ and /INT$_6$ interrupt inputs are actually supported by Commodore-Amiga as part of the Zorro II specification; the A2000 hardware did not provide the to software the required support mechanisms for the safe use of these lines. Each of these interrupt lines are shared by wired ORing, thus each line must be driven by an open-collector or equivalent output type, and all are pulled high by passive backplane resistors.

2.3.4 Slot Control Signals

This group of signals is responsible for the control of things that happen between expansion slots.

Slave (/SLAVEN)

Each slot has its own /SLAVE output, driven actively, all of which go into the collision detect circuitry. The "N" refers to the expansion slot number of the particular /SLAVE signal. Whenever a Zorro II PIC is responding to an address on the bus, it must assert its /SLAVE output within 35ns of /AS asserted. The /SLAVE output must be negated at the end of a cycle within 50ns of /AS negated. Late /SLAVE assertion on a Zorro II bus can result in loss of data setup times and other problems. A late /SLAVE negation for Zorro II cards can cause a collision to be detected on the following cycle. While the Zorro III sloppy cycle logic eliminates this fatal condition, late /SLAVE negation can nonetheless slow system performance unnecessarily. If more than one /SLAVE output occurs for the same address, or if a PIC asserts its /SLAVE output for an address reserved by the local bus, a collision is registered and results in /BERR being asserted.

Configuration Chain (/CFGINN, /CFGOUTN)

The slot configuration mechanism uses the bus signals /CFGOUTN and /CFGINN, where "N" refers to the expansion slot number. Each slot has its own version of each, which make up the configuration chain between Slots. Each subsequent /CFGIN is a result of all previous /CFGOUTs, going from slot 0 to the last slot on the expansion bus. During the AUTOCONFIG® process, an unconfigured Zorro PIC responds to the 64K address space starting at $00E80000 if its /CFGIN signal is asserted. All unconfigured PICs start up with /CFGOUT negated. When configured, or told to "shut up", a PIC will assert its /CFGOUT, which results in the /CFGIN of the next slot being asserted. The backplane passes on the state of the previous /CFGOUT to the next /CFGIN for any slot not occupied by a PIC, so there's no need to sequentially populate the expansion bus slots.

Data Output Enable (DOE)

This signal is used by an expansion card to enable the buffers on the data bus. The main Zorro II use of this line is to keep PICs from driving data on the bus until any other device is completely off the bus and the bus buffers are pointing in the correct direction. This prevents any contention on the data bus.

2.3.5 DMA Control Signals

There are various signals on the expansion bus that coordinate the arbitration of bus masters. Native Zorro III bus masters use some of the same logical signals, but their arbitration protocol is considerably different.

PIC is DMA Owner (/OWN)

This signal is asserted by an expansion bus DMA device when it becomes bus master. This output is to be treated as a wired-OR output between all expansion slots, any of which may have a PIC signalling bus mastership. Thus, this should be driven with an open-collector or similar output by any PIC using it. This signal is the main basis for data direction calculations between the local and expansion busses, and is pulled up by a backplane resistor.

Slot Specific Bus Arbitration (/BRN, /BGN)

These are the slot-specific /BRN and /BGN signals, where "N" refers to the expansion slot number. The bus request from each board is taken in by the bus controller and ultimately used to take over the system from 680x0 on the local bus. The bus controller eventually returns one bus grant to the winner among all requesting PICs. From the point of view of the individual PIC, the protocol is very similar to that of the 68000 arbitration mechanism. The PIC asserts /BRN on the rising edge of 7M; some time later, /BGN is returned on the falling edge of 7M. The PIC waits for all bus activity to finish, asserts /OWN followed by /BGACK, then negates /BRN, assuming bus mastership. It retains mastership until it negates /BGACK followed by /OWN.

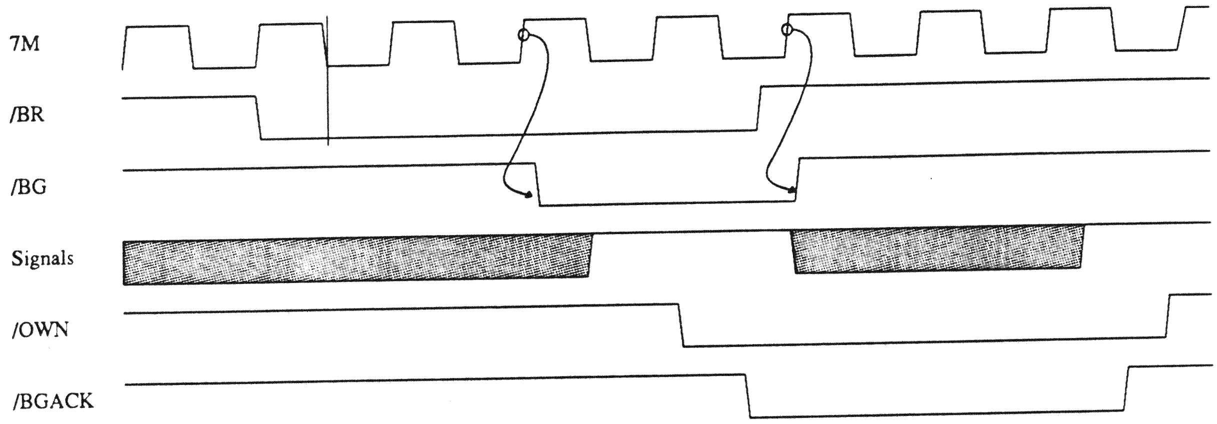


Figure 2-3: Zorro II Bus Arbitration

Bus Grant Acknowledge (/BGACK)

Any Zorro II PIC that receives a bus grant asserts this signal as long as it maintains bus mastery. This signal may never be asserted until the bus grant has been received, /AS is negated, /DTACK is negated, and /BGACK itself is negated, indicating that all other potential bus masters have relinquished the bus. This output is driven as a wired-OR output, so all PICs must drive it with an open collector or equivalent device, and a passive pullup is supplied by the backplane.

Bus Want/Clear (/GBG) ≡ (/BCLR) for Zorro III

This signal is asserted by the bus controller to indicate that a PIC wants to master the bus. A bus master assumes that the host CPU wants the bus, and that any time wasted as master is stealing time from the CPU. To avoid such waste, a master should use cache or FIFO to grab slow-coming data, and then transfer it all at once. /BCLR is asserted to indicate that additionally, another PIC wants the bus, and the current bus master should get off as soon as possible. This signal is equivalent to /GBG on the A2000 bus.

2.3.6 Addressing and Control Signals

These signals are various items used for the addressing of devices in Zorro II mode by the local bus and any expansion DMA devices. Most of these signals are very much like 68000 generated bus signals bi-directionally buffered to allow any DMA device on the bus to drive the local bus when such a device is the bus master.

Read Enable (READ)

This is the read enable for the bus, which is equivalent to the 68000's R/W output. READ asserted during a bus cycle indicates a read cycle, READ negated indicates a write cycle. Note that this signal may become valid in a cycle earlier than a 68000 R/W line would, but it remains valid at least as long at the cycle's end.

Address Bus (A_1-A_{23})

This is logically equivalent to the 68000's address bus, providing 16 megabytes of address space, although much of that space is not assigned to the expansion bus (see the memory map in *Figure 1-1*).

Address Strobe (/AS) ≡ (/CCS) for Zorro III

This is equivalent to the 68000 /AS, called /CCS, for Compatibility Cycle Strobe, in the Zorro III nomenclature. The falling edge of this strobe indicates that addresses are valid, the READ line is valid, and a Zorro II cycle is starting. The rising edge signals the end of a Zorro II bus cycle, signaling the current slave to negate all slave-driven signals as quickly as possible. Note that /CCS, like /AS, can stay asserted during a read-modify-write access over multiple cycle boundaries. To correctly support such cycles, a device must consider both the state of /CCS and the state of the data strobes. Many current Zorro II cards don't correctly support this 680x0 style bus lock.

Data Bus (D_0-D_{15})

This is a buffered version of the 680x0 data bus, providing 16 bits of data accessible by word or either byte. A PIC uses the DOE signal to determine when the bus is to be driven on reads, and the data strobes to determine when data is valid on writes.

Data Strobes (/UDS, /LDS) ≡ (/DS_3, /DS_2) for Zorro III

These strobes fall on data valid during writes, and indicate byte select for both reads and writes. The lower strobe is used for the lower byte (even byte), the upper strobe is used for the upper byte (odd byte). There is one slight difference between these lines and the 68000 data strobes. On reads of Zorro II memory space, both /DS_3 and /DS_2 will be asserted, no matter what the actual size of the requested transfer is. This is required to support caching of the Zorro II memory space. For Zorro II I/O space, these strobes indicate the actual, requested byte enables, just as would a 68000 bus master.

Data Transfer Acknowledge (/DTACK)

This signal is used to normally terminate both Zorro bus cycles. For Zorro II modes, it is equivalent to the 68000's Data Transfer Acknowledge input. It can be asserted by the bus slave during a Zorro II cycle at any time, but won't be sampled by the bus master until the falling edge of the S_4 state on the bus. Data will subsequently be latched on the S_6 falling edge after this, and the cycle terminated with /AS negated during S_7. If a Zorro II slave does nothing, this /DTACK will be driven by the bus controller with no wait states, making the bus essentially a 4 cycle synchronous bus. Any slow device on the bus that needs wait states has two options. It can modify the automatic /DTACK negating XRDY to hold off /DTACK. Alternately, it may assert /OVR to inhibit the bus controller's generation of /DTACK, allowing the slave to create its own /DTACK. Any /DTACK supplied by a slave must be driven with an open-collector or similar type output; the backplane provides a passive pullup.

Processor Status (FC_0-FC_2)

These signals are the cycle type or memory space bits, equivalent for the most part with the 68000 Processor Status outputs. They function mainly as extensions to the bus address, indicating which type of access is taking place. For Zorro II devices, any use of these lines must be gated with /BGACK, since they are not driven valid by Zorro II bus masters. However, when operating on the Zorro III backplane, Zorro II masters that don't drive the function codes will be seen generating an $FC_1 = 0$, which results in a valid memory access. Zorro II cycles are not generated for invalid memory spaces when the CPU is the bus master.

/DTACK Override (/OVR)

This signal is driven by a Zorro II slave to allow that slave to prevent the bus controller's /DTACK generation. This allows the slave to generate its own /DTACK. The previous use of this line to disable motherboard memory mapping, which was unsupported on the A2000 expansion bus, has now been completely removed. The use of XDRY or /OVR in combination with /DTACK is completely up to the board designer -- both methods are equally valid ways for a slave to delay /DTACK. In Zorro III mode, this pin is used for something completely different.

External Ready (XRDY)

This active high signal allows a slave to delay the bus controller's assertion of /DTACK, in order to add wait states. XRDY must be negated within 60ns of the bus master's assertion of /AS, and it will remain negated until the slave wants /DTACK. The /DTACK signal will be asserted by the bus controller shortly following the assertion of XRDY, providing the bus cycle is a S_4 or later. XRDY is a wired-OR from all PICs, and as such, must be driven by an open collector or equivalent output. In Zorro III mode, this pin is used for something completely different.

CHAPTER 3

BUS ARCHITECTURE

> *"We follow in the steps of our ancestory, and that cannot be broken."*
>
> *-Midnight Oil*

While the Zorro II bus design was based in a large part on an already existing bus cycle, the 68000 cycle, the Zorro III bus design had a much different set of preconditions. It is not modeled after any particular CPU specific bus protocol, but instead it's a logical outgrowth of both the need to support Zorro II cards on the same bus and the need to achieve various modern feature and preformance goals. These goals were summarized in Chapter 1, now they'll be covered in greater detail here.

3.1 Basic Zorro III Bus Cycles

The basic Zorro III bus cycle is a multiplexed address/data cycle which supplies a full 32 bits worth of address and data per simple cycle. The cycle is a fully asynchronous cycle. The bus master for a given cycle supplies strobes to indicate when address is valid, write data is valid, and read data may be driven. In return, the bus slave for a cycle supplies a strobe to indicate that it is responding to a bus address, and a strobe to indicate that it is done with the bus data for a write cycle, or has supplied valid bus data for a read cycle. The minimum theoretical bus speed is governed only by setup and hold time requirements for the various bus signals. Actual bus speeds is always a function of the bus master and bus slave active for a given cycle. This is considerably different than the way things work under the Zorro II bus, and for several good reasons, which are explained below.

3.1.1 Design Goals

For any computer bus, there are two basic possibilities concerning the fundamental operation of the bus; it's either synchronous or asynchronous. The difference is simple -- the synchronous bus is ultimately tied to a clock of some sort, while the asynchronous bus has no defined relationship to any clock signal. While Motorola specifies the 68000 bus cycle as an asynchronous cycle, they're really refering to the fact that most 68000 inputs are internally synchronized with the bus clock, and therefore, synchronous setup times on the bus do not have to be met to avoid metastability. But the 68000 bus, and the Zorro II bus by extension, are synchronous buses, based on a single bus clock (called E7M on the Zorro II bus). Most Zorro II signals are asserted relative to an edge of the bus clock, and most Zorro II inputs are sampled on an edge of the bus clock. The minimum Zorro II cycle is four bus clocks long, and every wait state added, regardless of the method, will result in a single additional bus clock wait, regardless of the asynchronous appearance of the termination and wait signals on the Zorro II bus.

The Zorro III bus is a fully asynchronous bus, in that all bus events are driven by strobes, and there is no reference clock. The choice of an asynchronous versus a synchronous bus design is governed by the intended application of the bus. Synchronous designs are preferred when a CPU and a memory system (eg, master and slave) can be very tightly coupled to each other. Such designs generally require a tight adherence to timing based on the specific CPU. This is optimal for tightly coupled systems, such the fast memory on the A3000 local bus. Synchronous designs can also be easier to do accurately, as the designer can use clock edges for scheduling events, and there's never any need to waste time in synchronizers to achieve a reliable design.

The design goals for an expansion bus are considerably different. While a fast memory circuit on a system motherboard can change for every new and better design, it's not feasable to require redesign of any significant number of expansion cards every time an improved motherboard design is created. And while a synchronous transfer can be optimal for matched clocks, it can be very inefficient for mismatched CPU and expansion clocks, as synchronizer delays must be introduced for any reliable operation. The A3000 project started with the need to support CPU systems at 16MHz and at 25MHz, and it's obvious that the growth of CPU clock speed will be here for some time to come. Zorro III cards are based on asynchronous handshaking between master and slave in both directions. This means that, as long as masters and slaves manage their own needs, any slave can work with any master. But as masters and slaves improve with technology, bus transfer speeds can automatically increase, without rendering any slower cards obsolete. The Zorro III bus attempts to address the needs of device expansion as much as the needs of memory expansion.

3.1.2 Simple Bus Cycle Operation

The normal Zorro III bus cycle is quite different than the Zorro II bus in many respects. *Figure 3-1* shows the basic cycle. There is no bus clock visible on the expansion bus; the standard Zorro II clocks are still active during Zorro III cycles, but they have no relationship to the Zorro II bus cycle. Every bus event is based on a relationship to a particular bus strobe, and strobes are alternately supplied by master and slave.

A Zorro III cycle begins when the bus master simultaneously drives addressing information on the address bus and memory space codes on the FC$_N$ lines, quickly following that with the assertion of the Full Cycle Strobe, /FCS; this is called the *address phase* of the bus. Any active slaves will latch the bus address on the falling edge of /FCS, and the bus master will tri-state the addressing information very shortly after /FCS is asserted. It's necessary only to latch A$_{31}$-A$_8$; the low order A$_7$-A$_2$ addresses and FC$_N$ codes are non-multiplexed.

As quickly as possible after /FCS is asserted, a slave device will respond to the bus address by asserting its /SLAVE$_N$ line, and possibly other special-purpose signals. The autoconfiguration process assigns a unique address range to each PIC base on its needs, just as on the Zorro II bus. Only one slave may respond to any given bus address; the bus controller will generate a /BERR signal if more than one slave responds to an address, or if a single slave responds to an address reserved for the local bus (this is called a bus collision, and should never happen in normal operation). Slaves don't usually respond to CPU memory space or other reserved memory space types, as indicated by the memory space code on the FC$_N$ lines (see Chapter 4 for details)!

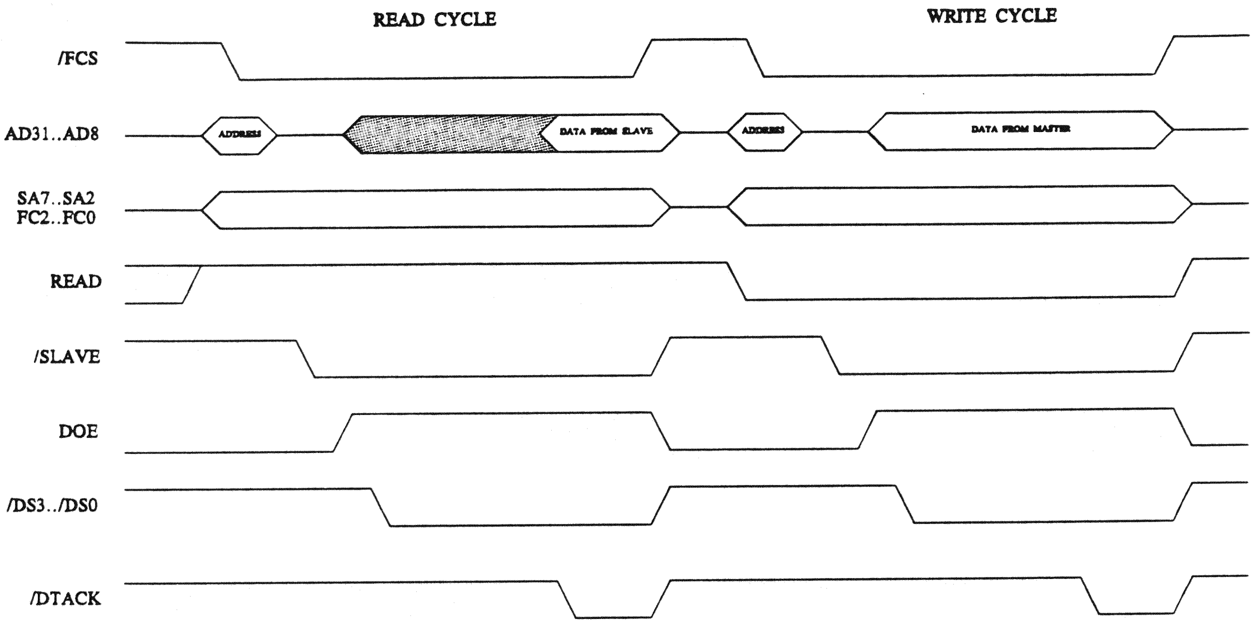


Figure 3-1: Basic Zorro III Cycles

The *data phase* is the next part of the cycle, and it's started when the bus master asserts DOE onto the bus, indicating that data operations can be started. The strobes are the same for both read and write cycles, but of course the data transfer direction is different.

For a read cycle, the bus master drives at least one of the data strobes /DS$_N$, indicating the physical transfer size requested (however, cachable slaves must always supply all 32 bits of data). The slave responds by driving data onto the bus, and then asserting /DTACK. The bus master then terminates the cycle by negating /FCS, at which point the slave will negate its

/SLAVEN line and tri-state its data. The cycle is done at this point. There are a few actions that modify a cycle termination, those will be covered in later sections.

The write cycle starts out the same way, up until DOE is asserted. At this point, it's the master that must drive data onto the bus, and then assert at least one /DSN line to indicate to the slave that data is valid and which data bytes are being written. The slave has the data for its use until it terminates the cycle by asserting /DTACK, at which point the master can negate /FCS and tri-state its data at any point. For maximum bus bandwidth, the slave can latch data on the falling edge of the logically ORed data strobes; the bus master doesn't sample /DTACK until after the data strobes are asserted, so a slave can actually assert /DTACK any time after /FCS.

3.2 Advanced Mode Support Logic

The Zorro III bus provides support for some more advanced things that weren't generally handled correctly on the Zorro II bus. Amiga computers have traditionally been supporting things that the more mainstream personal computers haven't. High speed DMA transfers and expansion coprocessors such as the Bridge Cards have been with the Amiga since the early days, and high performance main system CPUs with cache memory are now becoming common. The Zorro II bus never properly or easily supported such devices; the Zorro III bus attempts to make support of cache and coprocessor both possible and relatively straightforward. Other new features are covered in later sections.

3.2.1 Bus Locking

The first advanced modification of the basic bus cycle is bus locking, via the /LOCK signal. Bus locking is a hardware convention that allows a bus master to guarantee several cycles will be atomic on the bus. This is necessary to support the sharing of special "mail-box" memory between a bus master and an alternate PIC-based processor; Bridge Cards are an example of this kind of device. The Zorro II bus itself supports bus locking via the 68000 convention. However, the 68000 style of bus locking is often difficult to implement, and support for it was often ignored in Zorro II designs, especially those not directly concerned with multiprocessor support.

The Zorro III mechanism involves no change to the basic bus cycle, other than the monitoring of this /LOCK signal, and as such is much more reasonable to support. The /LOCK signal is asserted by a bus master at address time and maintained across cycles to lock out shared memory coprocessors, allowing hardware backed semaphores to easily be used between such coprocessors. We expect multiprocessing will be a greater concern on the Zorro III bus than it is at present; video coprocessors, RISC devices, and special purpose processors for image processing or mathematics should find a comfortable home on the Zorro III bus.

3.2.2 Cache Support

The other advanced cycle modifier on the Zorro III bus is the cache inhibit line, /CINH. On the Zorro II bus, there was originally no caching envisioned, and therefore no real support for caching of Zorro II PICs. First in the A2630 and later in the Zorro III bus's emulation of Zorro II, conventions were adopted to permit caching of Zorro II cards. These conventions aren't perfect; MMU tables will sometimes have to supplant this geographic mapping. While Zorro III doesn't have any cache consistency mechanisms for managing caches between several caching bus masters, it does allow cards that absolutely must not be cached to assert a cache inhibit line, /CINH, on a per-cycle basis (asserted at slave time by a responding slave). This cache management is basically the lowest level of a cache management system, mainly useful for support of I/O and other devices that shouldn't be cached. Software will be required for the higher levels of cache management.

3.3 Multiple Transfer Cycles

The multiplexed address/data design of the Zorro III bus has some definite advantages. It allows Zorro III cards to use the same 100 pin connector as the Zorro II cards, which results in every bus slot being a 32 bit slot, even if there's an alternate connector in-line with any or all of the system slots; current alternate connectors include Amiga Video and PC-AT (now sometimes called ISA, for *Industry Standard Architecture*, now that it's basically beyond the control of IBM) compatible connectors. This design also makes implementation of the bus controller for a system such as the A3000 simpler. And it can result in lower cost for Zorro III PICs in many cases.

The main disadvantage of the multiplexed bus is that the multiplexing can waste time. The address access time is the same for multiplexed and non-multiplexed buses, but because of the multiplexing time, Zorro III PICs must wait until *data time* to assert data, which places a fixed limit on how soon data can be valid. The Zorro III Multiple Transfer Cycle is a special mode designed to allow the bus to approach the speed of a non-multiplexed design. This mode is especially effective for high speed transfers between memory and I/O cards.

As the name implies, the Multiple Transfer Cycle is an extension of the basic full cycle that results in multiple 32 bit transfers. It starts with a normal full cycle *address phase* transaction, where the bus master drives the 32 bit address and asserts the /FCS signal. A master capable of supporting a Multiple Transfer Cycle will also assert /MTCR at the same time as /FCS. The slave latches the address and responds by asserting its /SLAVEN line. If the slave is capable of multiple transfers, it'll also assert /MTACK, indicating to the bus master that it's capable of this extended cycle form. If either /MTCR or /MTACK is negated for a cycle, that cycle will be a basic full cycle.

Assuming the multiple transfer handshake goes through, the multiple cycle continues to look similar to the basic cycle into the data phase. The bus master asserts DOE (possibly with write data) and the appropriate /DSN, then the slave responds with /DTACK (possibly with read data at the same time), just as usual. Following this, however, the cycle's character changes. Instead of terminating the cycle by negating /FCS, /DSN, and DOE, the master negates /DSN and /MTCR, but maintains /FCS and DOE. The slave continues to assert /SLAVEN, and the bus goes into what's called a *short cycle*.

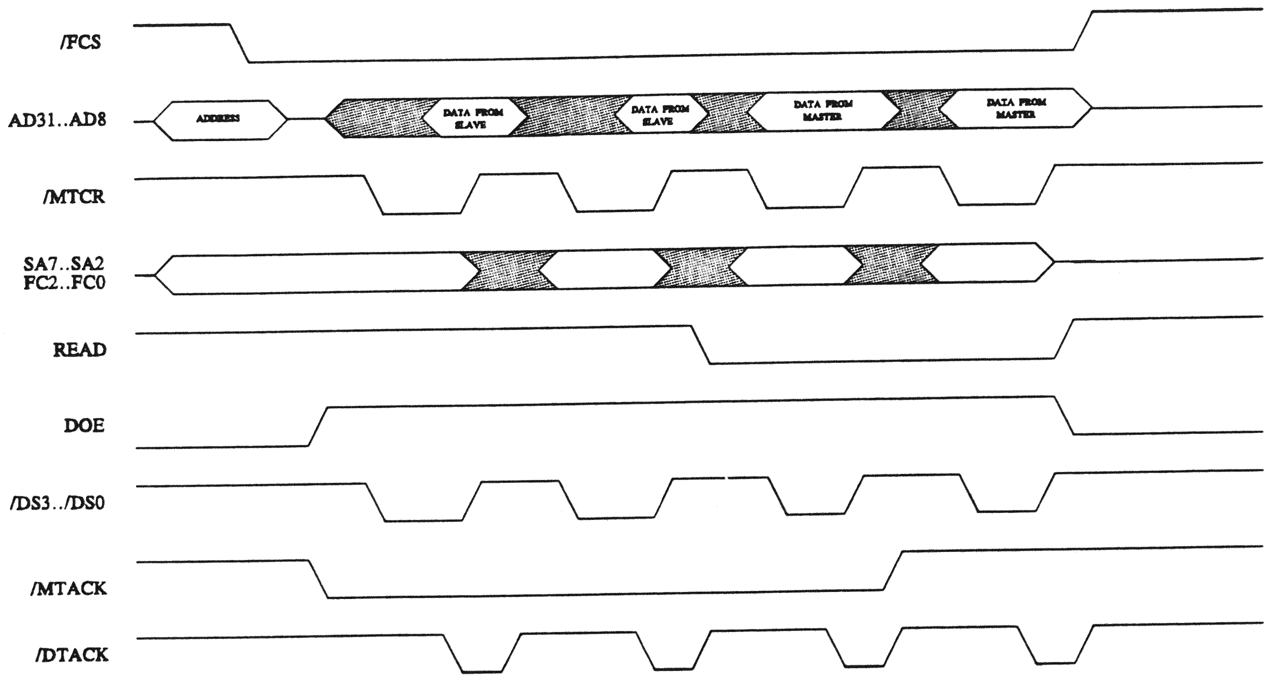


Figure 3-2: Multiple TransferCycles

The short cycle begins with the bus master driving the low order address lines A7-A2; these are the non-multiplexed addresses and can change without a new *address phase* being required (this is essentially a page mode, fully random accesses on this 256 byte page). The READ line may also change at this time. The master will then assert /MTCR to indicate to the slave that the short cycle is starting. For reads, the appropriate /DSN are asserted simultaneously with /MTCR, for writes, data and /DSN are asserted slightly after /MTCR. The slave will supply data for reads, then assert /DTACK, and the bus will will terminate the short cycle and start into either another short cycle or a full cycle, depending on the multiple cycle handshaking that has taken place.

The question of whether a subsequent cycle will be a full cycle or a short cycle is answered by multiple cycle arbitration. If the master can't sustain another short cycle, it will negate /FCS and DOE along with /MTCR at the end of the current short cycle, terminating the full cycle as well. The master always samples the state of /MTACK on the falling edge of /MTCR. If a slave can't support additional short cycles, it negates /MTACK one short cycle ahead of time. On the following short cycle, the bus master will see that no more short cycles can be handled by the slave, and fully terminate the multiple transfer cycle once this last short cycle is done.

PICs aren't absolutely required to support Multiple Transfer Cycles, though it is a highly recommended feature, especially for memory boards. And of course, all PICs must act intelligently about such cycles on the bus; a card doesn't request or acknowledge any Multiple Transfer Cycle it can't support.

3.4 Quick Bus Arbitration

The Zorro II bus does an adequate job of supporting multiple bus masters, and the Zorro III bus extends this somewhat by introducing fair arbitration to Zorro II cards. However, some desirable features cannot be added directly to the Zorro II arbitration protocol. Specifically, Zorro III bus arbitration is much faster than the Zorro II style, it prohibits bus hogging that's possible under the Zorro II protocol, and it supports intelligent bus load balancing.

Load balancing requires a bit of explanation. A good analogy is to that of software multitasking; there, an operating system attempts to slice up CPU time between all tasks that need such time; here, a bus controller attempts to slice up bus time between all masters that need such time. With preemptive multitasking such as in the Amiga and UNIX OSs, equal CPU time can be granted to every task (possibly modified by priority levels), and such scheduling is completely under control of the OS; no task can hog the CPU time at the expense of all others. An alternate multitasking scheme is a popular add-on to some originally non-multasking operating systems lately. In this scheme, each task has the CPU until it decides to give up the CPU, basically making the effectiveness of the CPU sharing at the mercy of each task. This is exactly the same situation with masters on the Zorro II bus. The Zorro III arbitration mechanism attempts to make bus scheduling under the control of the bus controller, with masters each being scheduled on a cycle-by-cycle basis.

When a Zorro III PIC wants to master the bus, it *registers* with the bus controller. This tells the bus controller to include that PIC in its scheduling of the expansion bus. There may be any number of other PICs registered with the bus controller at any given time. The CPU is always scheduled expansion bus time, and other local bus devices, such as a hard disk contoller, may be registered from time to time.

Once registered, a PIC sits idle until it receives a *grant* from the bus controller. A grant is permission from the bus controller that allows the PIC to master the Zorro III bus for one full cycle. A PIC always gets one full cycle of bus time when given a grant, and assuming it stays registered, it may receive additional full cycles. Within the full cycle, the PIC may run any number of Multiple Transfer Cycles, assuming of course the responding slave supports such cycles. For multiprocessor support, a PIC will be granted multiple atomic full cycles if it locks the bus. This feature is only for support of hardware semaphores and other such multiprocessor needs; it is not intended as a means of bus hogging!

Figure 3-3 shows the basics of Zorro III bus arbitration, which is pretty simple. While it uses some of the same signals as the 680x0 inspired Zorro II bus arbitration mechanism, it has nothing to do with 680x0 bus arbitration; the /BRN and /BGN signals should be thought of as completely new signals. In order to register with the bus controller as a bus master, a PIC asserts its private /BRN strobe on the rising edge of the 7M clock, and negates it on the next rising edge. The bus controller will indicate mastership to a registered bus master by asserting its /BGN. Once granted the bus, the PIC drives only the standard cycle signals: addresses, /FCS, /EDSN, data, etc. in a full cycle. The bus controller manages the assertion of /OWN and /BGACK, which are important only for bus management and Zorro II support. While a scheduling scheme

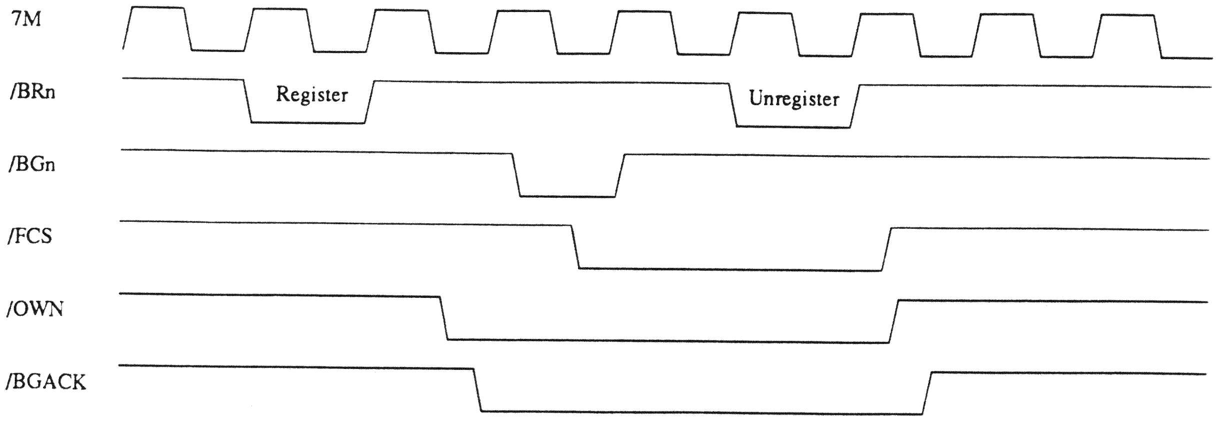


Figure 3-3: Zorro III Bus Arbitration

isn't part of this bus specification, the bus master will only be guaranteed one bus cycle at a time. The /BGN line is negated shortly after the master asserts /FCS unless the bus controller is planning to grant multiple full cycles to the master. The only thing that'll force the controller to grant multiple full cycles is a locked bus. Any master that works better with multiple cycles, such as devices with buffers to empty into memory, should run a Multiple Transfer Cycle to transfer several longwords during the same full cycle. For this reason, slave cards are encouraged to support Multiple Transfer Cycles, even if they don't necessarily run any faster during them.

Once a registered bus master has no more work to do, it unregisters with the bus controller. This works just like registering -- the PIC asserts /BRN on the rise of 7M, then negates it on next rising 7M. This is best done during the last cycle the bus master requires on the bus. If a registered master gets a grant before unregistering and has no work to do, it can unregister without asserting /FCS, to give back the bus without runing a cycle. It's always far better to make sure that the master unregisters as quickly as possible. Bus timeout causes an automatic unregistering of the registered master that was granted that timed-out cycle; this guarantees that an inactive registered master can't drag down the system. If a master sees a /BERR during a cycle, it should terminate that cycle immediately and re-try the same cycle. If the retried cycle results in a /BERR as well, nothing more can be done in hardware; notification of the driver program is the usual recourse.

The bus controller may have to mix Zorro II style bus arbitration in with Zorro III arbitration, as Zorro II and Zorro III cards can be freely mixed in a backplane. Because of this, Multiple Transfer Cycles, and the self-timed nature of Zorro III cards, there's no way to guarantee the latency between bus grants for a Zorro III card. The bus controller does, however, make sure that all masters are fairly scheduled so that no starvation occurs, if at all possible. Zorro III cards must use Zorro III style bus arbitration; although current Zorro III backplanes can't differentiate between Zorro II and Zorro III cards when they request (other than by the request mechanism), it can't be assumed that a backplane will support Zorro III cycles with Zorro II mastering, or visa-versa.

3.5 Quick Interrupts

While the Zorro II bus has always supported shared interrupts, the Zorro III bus supports a mechanism wherein the interrupting PIC can supply its own vector. This has the potential to make such vectored interrupts much faster than conventional Zorro II chained interrupts, arbitrating the interrupting device in hardware instead of software.

A PIC supporting quick interrupts has on-board registers to store one or more vector numbers; the numbers are obtained from the OS by the device driver for the PIC, and the PIC/driver combination must be able to handle the situation in which no additional vectors are available. During system operation, this PIC will interrupt the system in the normal manner, by asserting one of the bus interrupt lines. This interrupt will cause an interrupt vector cycle to take place on the bus. This cycle arbitrates in hardware between all PICs asserting that interrupt, and it's a completely different type of Zorro III cycle, as illustrated in *Figure 3-4*.

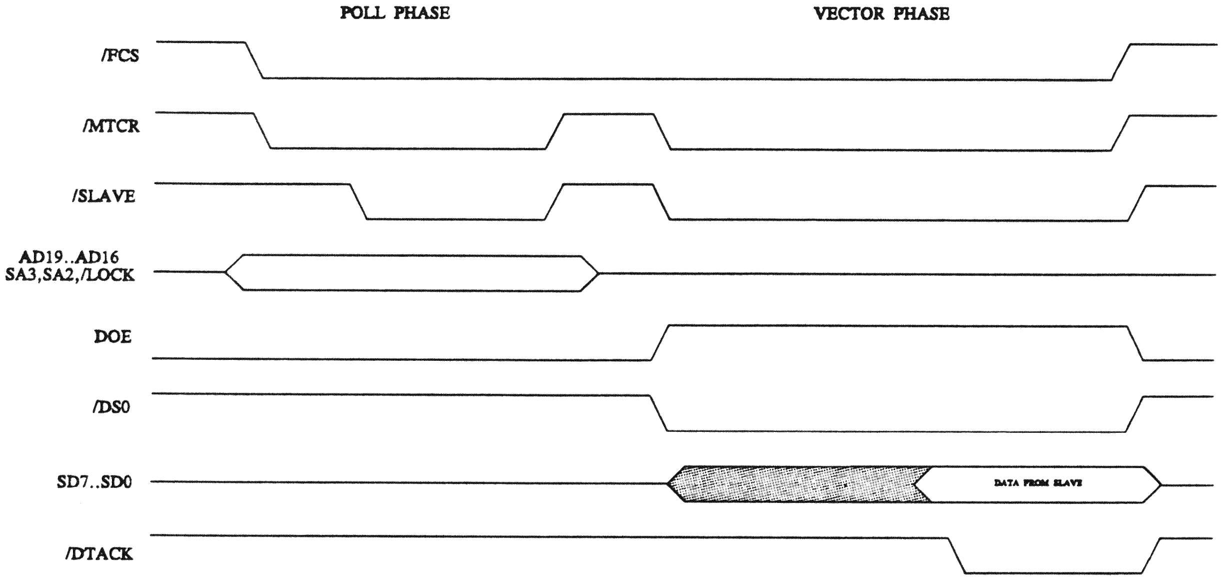


Figure 3-4: Interrupt Vector Cycle

The bus controller will start an interrupt vector cycle in response to an interrupt asserted by any PIC. This cycle starts with /FCS and /MTCR asserted, a FC code of 7 (CPU space), a CPU space cycle type, given by address lines A_{16}-A_{19}, of 15, and the interrupt number, which is on A_1-A_3 (A_1 is on the /LOCK line, as in Zorro II cycles). The interrupt numbers 2 and 6 are currently defined, corresponding to /INT$_2$ and /INT$_6$ respectively; all others are reserved for future use. At this point, called the *polling phase*, any PIC that has asserted an interrupt and wants to supply a vector will decode the FC lines, the cycle type, match its interrupt number against the one on the bus, and assert /SLAVE$_N$ if a match occurs. Shortly thereafter, the /MTCR line is negated, and the slaves all negate /SLAVE$_N$. But the cycle doesn't end.

The next step is called the *vector phase*. The bus controller asserts one /SLAVE$_N$ back to one of the interrupting PICs, along with /MTCR and /DS$_0$, but no addresses are supplied. That

PIC will then assert its 8 bit vector onto the logical D_0-D_7 (physically AD_{15}-AD_8) of the 32 bit data bus and /DTACK, as quickly as possible, thus terminating the cycle. The speed here is very critical; an automatic autovector timeout will occur very quickly, as any actual waiting that's required for the quick interrupt vector is potentially delaying the autovector response for Zorro II style interrupts. A PIC stops driving its interrupt when it gets the response cycle; it must also be possible for this interrupt to be cleared in software (eg, the PIC must make choice of vectoring vs. autovectoring a software issue).

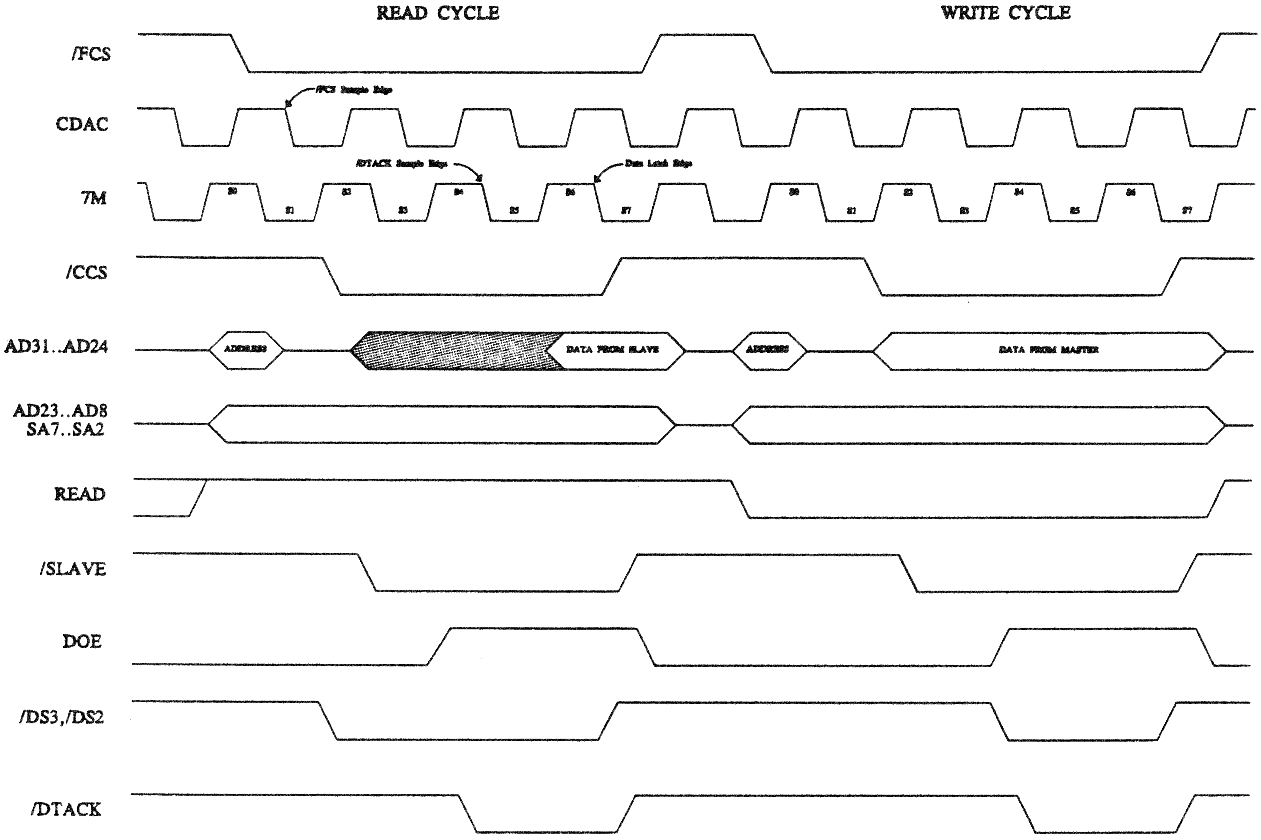

READ CYCLE
WRITE CYCLE
/FCS
CDAC
/FCS Sample Edge
7M
/DTACK Sample Edge
Data Latch Edge
S0
S1
S2
S3
S4
S5
S6
S7
S0
S1
S2
S3
S4
S5
S6
S7
/CCS
AD31..AD24
ADDRESS
DATA FROM SLAVE
ADDRESS
DATA FROM MASTER
AD23..AD8
SA7..SA2
READ
/SLAVE
DOE
/DS3,/DS2
/DTACK

Figure 3-5: Zorro II Within Zorro III

3.6 Compatibility with Zorro II Devices

As detailed in Chapter 2, the Zorro III bus supports a bus cycle mode very similar to the 68000-based Zorro II bus, and is expected to be compatible with all properly designed Zorro II PICs. As shown in *Figure 1-1*, Zorro II and Zorro III expansion spaces are geographically mapped on the Zorro III bus. The mapping logic resides on the bus, and operates on the bus address presented for any cycle. Every cycle starts out assuming a Zorro III cycle, but the mapping logic will inscribe a Zorro II cycle within the Zorro III cycle if the address range is right. *Figure 3-5* details the bus action for this mode.

The cycle starts out with the usual address phase activity; the bus master asserts /FCS after asserting the full 32 bit address onto the address bus. The bus decoder maps the bus address asynchronously and quickly, so that by the time /FCS is asserted, the memory space is determined. A Zorro II space access will cause A_8-A_{23} to remain asserted, rather than being tri-stated along with A_{24}-A_{31}, as the Zorro III cycle normally does. The bus controller synchs the asynchronous /FCS on the falling edge of CDAC, then drives /CCS (the /AS equivalent) out on the rising edge of 7M, based on that synched /FCS. For a read cycle, /DS_3 and/or /DS_2 (the /UDS and /LDS replacements, respectively) would be asserted along with /CCS; write cycles see those lines asserted on the next rising edge of 7M, at S_4 time. The DOE line is also asserted at the start of S_4.

The bus controller starts to sample /DTACK on the falling edge of 7M between S_4 and S_5, adding wait states until /DTACK is encountered. As per Zorro II specs, the PIC need not create a /DTACK unless it needs that level of control; there are Zorro II signals to delay the controller-generated /DTACK, or take it over when necessary. The controller will drive its automatic /DTACK at the start of S_4, leaving plenty of time for the sampling to come at S_5. Once a /DTACK is encountered, cycle termination begins. The controller latches data on the falling 7M edge between S_6 and S_7, and also negates /CCS and the /DS_N at this time. Shortly thereafter, the controller negates /DTACK (when controlling it), DOE, and tri-states the data bus, getting ready for the next cycle.

CHAPTER 4

SIGNAL DESCRIPTION

> *"Pushing back the limits of human achievement, reaching for the stars, that's not something we do. It's what we are."*
>
> *-Michael Swaine*

The signals detailed here are the Zorro III mode signals. While some of this information is the same in as the Zorro II signal description of Chaper 2, many like-seeming bus signals behave differently in Zorro III mode than Zorro II mode. These can be a very important differences; thus the complete set of signals is detailed here.

4.1 Power Connections

The expansion bus provides several different voltages designed to supply expansion devices. These are basically the same for the Zorro III bus as they were for the Zorro II bus, with the exception of one pin, and that the specification has been clarified a bit. Note that all Zorro III PICs must list their power consumption specifications.

Digital Ground (Ground)
This is the digital supply ground used by all expansion cards as the return path for all expansion supplies.

Main Supply (+5VDC)
This is the main power supply for all expansion cards, and it is capable of sourcing large currents; each PIC can draw up to 2.0 Amps @ +5VDC.

Negative Supply (-5VDC)

This is a negative version of the main supply, for small current loads only; each PIC can draw up to 60 mA @ -5VDC.

High Voltage Supply (+12VDC)

This is a higher voltage supply, useful for communications cards and other devices requiring greater that digital voltage levels. This is intended for relatively small current loads only; each PIC can draw up to 500mA @ +12VDC.

Negative High Supply (-12VDC)

Negative version of the high voltage supply, also used in communications applications, and similarly intended for small loads only; each PIC can draw up to 60 mA @ -12VDC.

4.2 Clock Signals

The expansion bus provides clock signals for expansion boards. The main use for these clocks on Zorro III cards is bus arbitration clocking. There is no relationship between any of these clocks and normal Zorro III bus activity. The relationship between these clocks is illustrated in *Figure 2-2*.

/C1 Clock

This is a 3.58 MHz clock (3.55 MHz on PAL systems) that's synched to the falling edge of the 7M system clock.

/C3 Clock

This is a 3.58 MHz clock (3.55 MHz on PAL systems) that's synched to the rising edge of the 7M system clock.

CDAC Clock

This is a 7.16 MHz system clock (7.09 MHz on PAL systems) which trails the 7M clock by 90° (approximately 35ns).

E Clock

This is the 68000 generated "E" clock, used for 6800 family peripherals driven by "E" and 6502 peripherals driven by Φ_2. This clock is four 7M clocks high, six clocks low, as per the 68000 spec.

7M Clock

This is the 7.16 MHz system clock (7.09 MHz on PAL systems). This clock drives the bus master registration mechanism for Zorro III bus masters.

4.3 System Control Signals

The signals in this group are available for various types of system control; most of these have an immediate or near immediate effect on expansion cards and/or the system CPU itself.

Hardware Bus Error/Interrupt (/BERR)

This is a general indicator of a bus fault or special condition of some kind. Any expansion card capable of detecting a hardware error relating directly to that card can assert /BERR when that bus error condition is detected, especially any sort of harmful hardware error condition. This signal is the strongest possible indicator of a bad situation, as it causes all PICs to get off the bus, and will usually generate a level 2 exception on the host CPU. For any condition that can be handled in software and doesn't pose an immediate threat to hardware, notification via a standard processor interrupt is the better choice. The bus controller will drive /BERR in the event of a detected bus collision or DMA error (an attempt by a bus master to access local bus resources it doesn't have valid access permission for). All cards must monitor /BERR and be prepared to tri-state all of their on-bus output buffers whenever this signal is asserted. An expansion bus master will attempt to retry a cycle aborted by a single /BERR and notify system software in the case of two subsequent /BERR results. Since any number of devices may assert /BERR, and all bus cards must monitor it, any device that drives /BERR must drive with an open collector or similar device, and any device that monitors /BERR should place a minimal load on it. This signal is pulled high by a passive backplane resistor.

Note that, especially for the slave device being addressed, that /BERR alone is not always necessaily an indication of a bus failure in the pure sense, but may indicate some other kind of unusual condition. Therefore, a device should still respond to the bus address, if otherwise appropriate, when a /BERR condition is indicated. It simply tri-states is bus buffers and other outputs, and waits for a change in the bus state. If the /BERR signal is negated with the cycle unterminated, the special condition has been resolved and the slave responds to the rest of the cycle as it normally would have. If the cycle is terminated by the bus master, the resolution of the special condition has indicated that the addressed slave is not needed, and so the cycle terminates without the slave being used.

System Reset (/RESET, /IORST)

The bus supplies two versions of the system reset signal. The /RESET signal is bidirectional and unbuffered, allowing an expansion card to hard reset the system. It should only be used by boards that need this reset capability, and is driven only by an open collector or similar device. The /IORST signal is a buffered output-only version of the reset signal that should be used as the normal reset input to boards not concerned with resetting the system on their own. All expansion devices are required to reset their autoconfiguration logic when /IORST is asserted. These signals are pulled high by passive backplane resistors.

System Halt (/HLT)

This signal is driven, along with /RESET, to assert a full-system reset. A full-system reset is asserted on a powerup reset or a keyboard reset; any PIC that needs to differentiate between full system and I/O reset should monitor /HLT and /IORST unless it also needs to drive a reset condition. This is driven with an open-collector output, or the equivalent, and pulled up by a backplane resistor.

System Interrupts

Two of the decoded, level sensitive 680x0 interrupt inputs are available on the expansion bus, and these are labelled as /INT2 and /INT6. Each of these interrupt lines is shared by wired ORing, thus each line must be driven by an open-collector or equivalent output type. Zorro III interrupts can be handled Zorro II style, via autovectors and daisy-chained polling, or they can be vectored using the quick interrupt protocol described in Chapter 3. Zorro II and Zorro III systems originally provided /INT1, /INT4, /INT5, and /INT7 lines as well, but as these were never properly supportable by system software, they have been eliminated, those lines now considered reserved for future use in a Zorro III system.

4.4 Slot Control Signals

This group of signals is responsible for the control of things that happen between expansion slots.

Slave (/SLAVEN)

Each slot has its own /SLAVEN output, driven actively, all of which go into the collision detect circuitry. The "N" refers to the expansion slot number of the particular /SLAVE signal. Whenever a Zorro III PIC is responding to an address on the bus, it must assert its /SLAVEN output very quickly. If more than one /SLAVEN output occurs for the same address, or if a PIC asserts its /SLAVEN output for an address reserved by the local bus, a collision is registered and the bus controller asserts /BERR. The bus controller will assert /SLAVEN back to the interrupting device selected during a Quick Interrupt cycle, so any device supporting Quick Interrupts must be capable of tri-stating its /SLAVEN; all others can drive SLAVEN with a normal active output.

Configuration Chain (/CFGINN, /CFGOUTN)

The slot configuration mechanism uses the bus signals /CFGOUTN and /CFGINN, where "N" refers to the slot number. Each slot has its own version of both signals, which make up the *configuration chain* between slots. Each subsequent /CFGINN is a result of all previous /CFGOUTs, going from slot 0 to the last slot on the expansion bus. During the autoconfiguration process, an unconfigured Zorro III PIC responds to the 64K address space starting at either $00E80000 or $FF000000 if its /CFGINN signal is asserted. All unconfigured PICs start up with /CFGOUTN negated. When configured, or told to "shut up", a PIC will assert its /CFGOUTN, which results in the /CFGINN of the next slot being asserted. Backplane logic automatically passes on the state of the previous /CFGOUTN to the next /CFGINN for any slot not occupied by a PIC, so there's no need to sequentially populate the expansion bus slots.

Backplane Type Sense (SenseZ3)

This line can be used by the PIC to determine the backplane type. It is grounded on a Zorro II backplane, but floating on a Zorro III backplane. The Zorro III PIC connects this signal to a 1K pullup resistor to generate a real logic level for this line. It's possible, though more complicated, to build a Zorro III PIC that can actually run in Zorro II mode when in a Zorro II backplane. It's hardly necessary or required to support this backward compatibility mechanism, and in many cases it'll be inpractical. The Zorro III specification does require that this signal be used, at least, to shut the card down and pass /CFGIN to /CFGOUT when in a Zorro II backplane.

4.5 DMA Control Signals

There are various signals on the expansion bus that coordinate the arbitration of bus masters. Zorro II bus masters use some of the same logical signals, but their arbitration protocol is considerably different.

PIC is DMA Owner (/OWN)

This is asserted by the bus controller when a master is about to go on the bus and indicates that some master owns the bus. Zorro II bus masters drive this, and some Zorro III slaves may find a need to monitor it, or /BGACK, to determine who's the bus master. This is ordinarily not important to Zorro III PICs, and they may not drive this line.

Slot Specific Bus Arbitration (/BRN, /BGN)

These are the slot-specific /BRN and /BGN signals, where "N" refers to the expansion slot number. The bus request from each board is taken in by the bus controller and ultimately used to take over the system from the primary bus master, which is always the local master. Zorro III PICs toggle /BRN to register or unregister as a master with the bus controller. /BGN is asserted to one registered PIC at a time, on a cycle by cycle basis, to indicate to the PIC that it gets the bus for one full cycle.

Bus Grant Acknowledge (/BGACK)

Asserted by the bus controller when a master is about to go on the bus. As with /OWN, most Zorro III PICs ignore this signal, and none may drive it.

Bus Want/Clear (/BCLR)

This signal is asserted by the bus controller to indicate that a PIC wants to master the bus; Zorro III cards can use this to determine if any Zorro II bus requests are pending; Zorro III bus requests don't affect /BCLR.

4.6 Address and Related Control Signals

These signals are various items used for the addressing of devices in Zorro III mode by bus masters either on the bus or from the local bus. The bus controller translates local bus signals (68030 protocol on the A3000) into Zorro III signals; masters are responsible for creating the appropriate signals via their own bus control logic.

Read Enable (READ)

Read enable for the bus; READ is asserted by the bus master during a bus cycle to indicate a read cycle, READ is negated to indicate a write cycle. READ is asserted at address time, prior to /FCS, for a full cycle, and prior to /MTCR for a short cycle. READ stays valid throughout the cycle; no latching required.

Multiplexed Address Bus (A_8-A_{31})

These signals are driven by the bus master during address time, prior to the assertion of /FCS. Any responding slave must latch as many of these lines as it needs on the falling edge of /FCS, as they're tri-stated very shortly after /FCS goes low. These addresses always include all configuration address bits for normal cycles, and the cycle type information for Quick Interrupt cycles.

Short Address Bus (A_2-A_7)

These signals are driven by the bus master during address time, prior to the assertion of /FCS, for full cycles, and prior to the assertion of /MTCR for short cycles. They stay valid for the entire full or short cycle, and as such do not need to be latched by responding slaves.

Table 4-1: Memory Space Type Codes

FC_0	FC_1	FC_2	Address Space Type	Z3 Response
0	0	0	Reserved	None
0	0	1	User Data Space	Memory
0	1	0	User Program Space	Memory
0	1	1	Reserved	None
1	0	0	Reserved	None
1	0	1	Supervisor Data Space	Memory
1	1	0	Supervisor Program Space	Memory
1	1	1	CPU Space	Interrupts

Memory Space (FC_0-FC_2)

The memory space bits are an extension to the bus address, indicating which type of access is taking place. Zorro III PICs must pay close attention to valid memory space types, as the space type can change the type of the cycle driven by the current bus master. The encoding is the same as the valid Motorola function codes for normal accesses. These are driven at address time, and like the low short address, are valid for an entire short or full cycle.

Compatibility Cycle Strobe (/CCS)

This is equivalent to the Zorro II address strobe, /AS. A Zorro III PIC doesn't use this for normal operation, but may use it during the autoconfiguration process if configuring at the Zorro II address. AUTOCONFIG® cycles at $00E8xxxx always look like Zorro II cycles, though of course /FCS and the full Zorro III address is available, so a card can use either Zorro II or Zorro III addressing to start the cycle. However, using the /CCS strobe can save the designer the need to compare the upper 8 bits of address. Data must be driven Zorro II style, though if the /DS_N lines are respected for reads, /CINH is asserted, and /MTACK is negated, the resulting Zorro III cycle will fit within the expected Zorro II cycle generated by the bus controller. Yes, that should sound weird; it's based on the mapping of Zorro II vs. Zorro III signals, and of course the fact that /FCS always starts any cycle. Also note that a bus cycle with /CCS asserted and /FCS negated is always a Zorro II PIC-as-master cycle. Many Zorro III cards will instead configure at the alternate $FF00xxxx base address, fully in Zorro III mode, and thus completely ignore this signal.

Full Cycle Strobe (/FCS)

This is the standard Zorro III full cycle strobe. This is asserted by the bus master shortly after addresses are valid on the bus, and signals the start of any kind of Zorro III bus cycle. Shortly after this line is asserted, all the multiplexed addresses will go invalid, so in general, all slaves latch the bus address on the falling edge of /FCS. Also, /BG$_N$ line is negated for a Zorro III mastered cycle shortly after /FCS is asserted by the master.

4.7 Data and Related Control Signals

The data time signals here manage the actual transfer of data between master and slave for both full and short cycle types. The burst mode signals are here too, as they're basically data phase signals even through they don't only concern the transfer of data.

Data Output Enable (DOE)

This signal is used by an expansion card to enable the buffers on the data bus. The bus master drives this line is to keep slave PICs from driving data on the bus until *data time*.

Data Bus (D$_0$-D$_{31}$)

This is the Zorro III data bus, which is driven by either the master or the slave when DOE is asserted by the master (based on READ). It's valid for reads when /DTACK is asserted by the slave; on writes when at least one of /DS$_N$ is asserted by the master, for all cycle types.

Data Strobes (/DS$_N$)

These strobes fall during *data time*; /DS$_3$ strobes D$_{24}$-D$_{31}$, while /DS$_0$ strobes D$_0$-D$_7$. For write cycles, these lines signal data valid on the bus. At all times, they indicate which bytes in the 32 bit data word the bus master is actually interested in. For cachable reads, all four bytes must be returned, regardless of the value of the sizing strobes. For writes, only those bytes corresponding to asserted /DS$_N$ are written. Only contiguous byte cycles are supported; e.g. /DS$_{3-0}$ = 2, 4, 5, 6, or 10 is invalid.

Data Transfer Acknowledge (/DTACK)

This signal is used to normally terminate a Zorro III cycle. The slave is always responsible for driving this signal. For a read cycle, it asserts /DTACK as soon as it has driven valid data onto the data bus. For a write cycle, it asserts /DTACK as soon as it's done with the data. Latching the data on writes may be a good idea; that can allow a slave to end the cycle before it has actually finished writing the data to its local memory.

Cache Inhibit (/CINH)

This line is asserted at the same time as /SLAVE$_N$ to indicate to the bus master that the cycle must not be cached. If a device doesn't support caching, it must assert /CINH and actually obey the /DS_N byte strobes for read cycles. Conversely, if the device supports caching, /CINH is negated and the device returns all four bytes valid on reads, regardless of the actual supplied /DS_N strobes.

Multiple Cycle Transfers (/MTCR,/MTACK)

These lines comprise the Multiple Transfer Cycle handshake signals. The bus master asserts /MTCR at the start of *data time* if it's capable of supporting Multiple Transfer Cycles, and the slave asserts /MTACK with /SLAVE_N if it's capable of supporting Multiple Transfer Cycles. If the handshake goes through, /MTCR strobes in the short address and write data as long as the full cycle continues.

CHAPTER 5

TIMING

"When dealing with the insane, the best method is
to pretend to be sane."

-Hermann Hesse

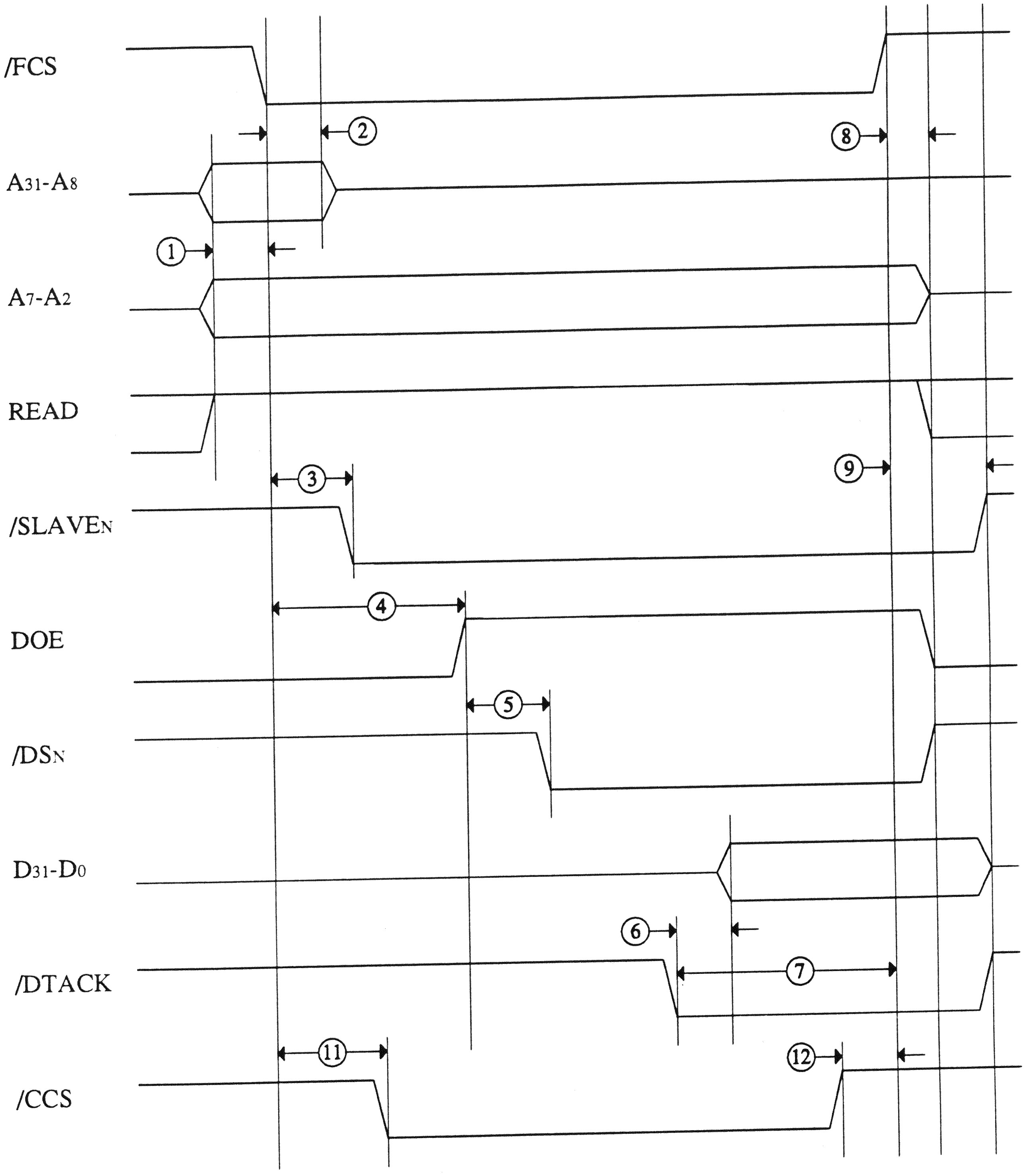
Some of this information is considered preliminary. Nothing is expected to get any more speed critical, but as mentioned previously, the testing of Zorro III designs has just started at the time of this writing, final bus controllers are not yet available, and only a few PIC designs have even been conceived.

This section covers the various timing specifications in detail for different Zorro III operations. It's important to realize that this timing information is a **specification**. Actual Zorro III systems may offer much more relaxed timings. Today. The whole point of the specification is that as long as all Zorro III PICs and all Zorro III backplanes base things on the timings given here, they'll always work together nicely. Any design based on the actual characteristics of any particular backplane will very likely wind up working only on that particular backplane.

The philosophy of timing on the Zorro III bus is to keep things as simple as possible without compromising the performance goals of the bus. Zorro III PICs are expected to be based on F-Series or ACT-series TTL logic, fast PALs, and possibly full custom chip designs. It's very unlikely the designer will meet any of these specifications with the LS parts left over from old Zorro II card designs.

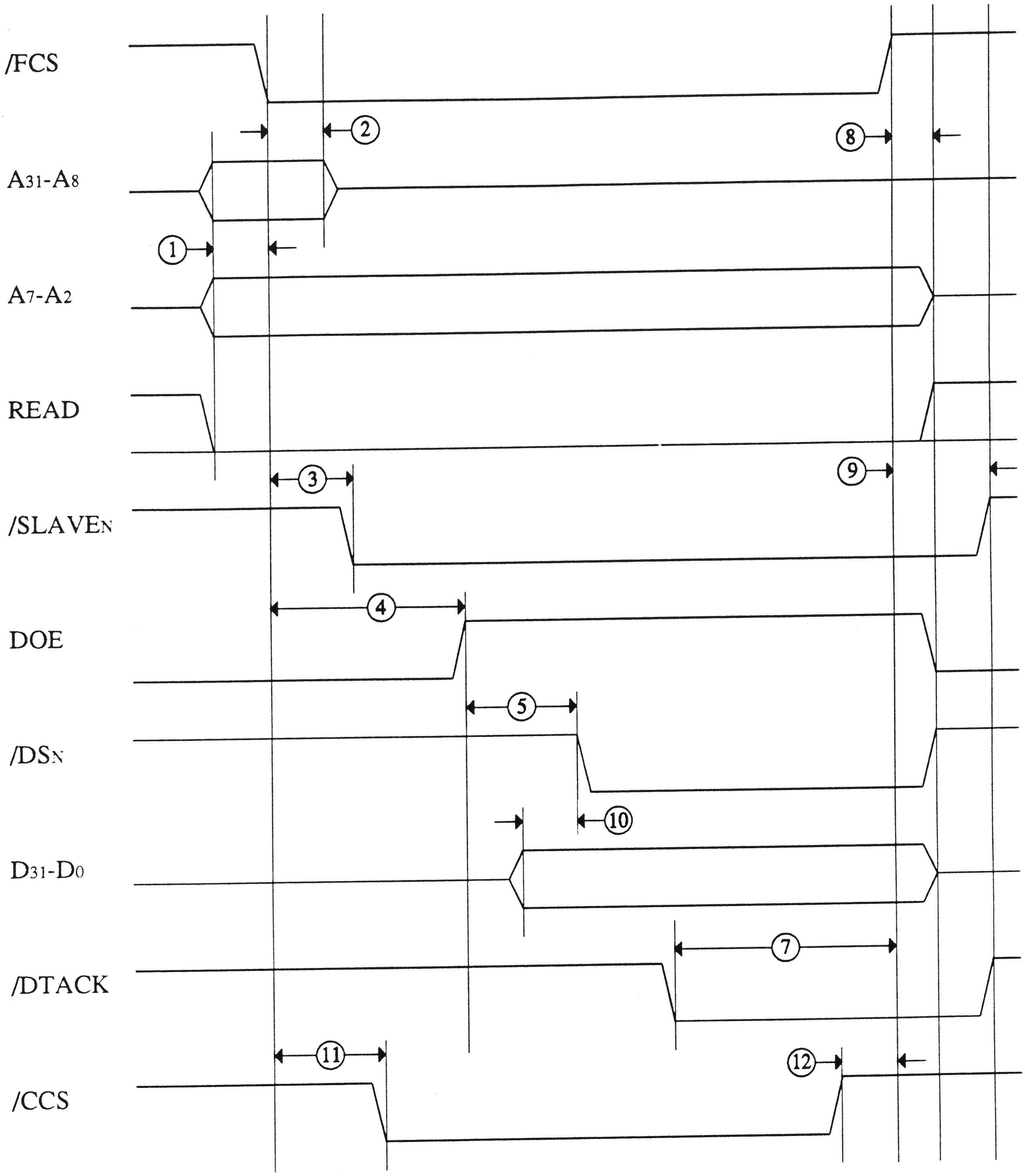
5.1 Standard Read Cycle Timing

No.	Name	Symbol	Min	Max
1	Address setup to /FCS	TAFS	15ns	-----
2	Address hold from /FCS	THAF	10ns	-----
3	/FCS to /SLAVEN delay	TSLV	-----	25ns
4	/FCS to DOE delay	TDOE	30ns	-----
5	DOE to /DSN delay	TDS	10ns	-----
6	Data setup to /DTACK	TRDS	0ns	-----
7	/DTACK to /FCS off	TOFF	10ns	-----
8	Master signal hold from /FCS off	THMC	0ns	5ns
9	Slave signal hold from /FCS off	THSC	0ns	15ns
11	/FCS to /CCS delay	TCCS	35ns	175ns
12	/CCS off to /FCS off	TOVL	40ns	-----


/FCS
A31-A8
A7-A2
READ
/SLAVEN
DOE
/DSN
D31-D0
/DTACK
/CCS
①
②
③
④
⑤
⑥
⑦
⑧
⑨
⑪
⑫

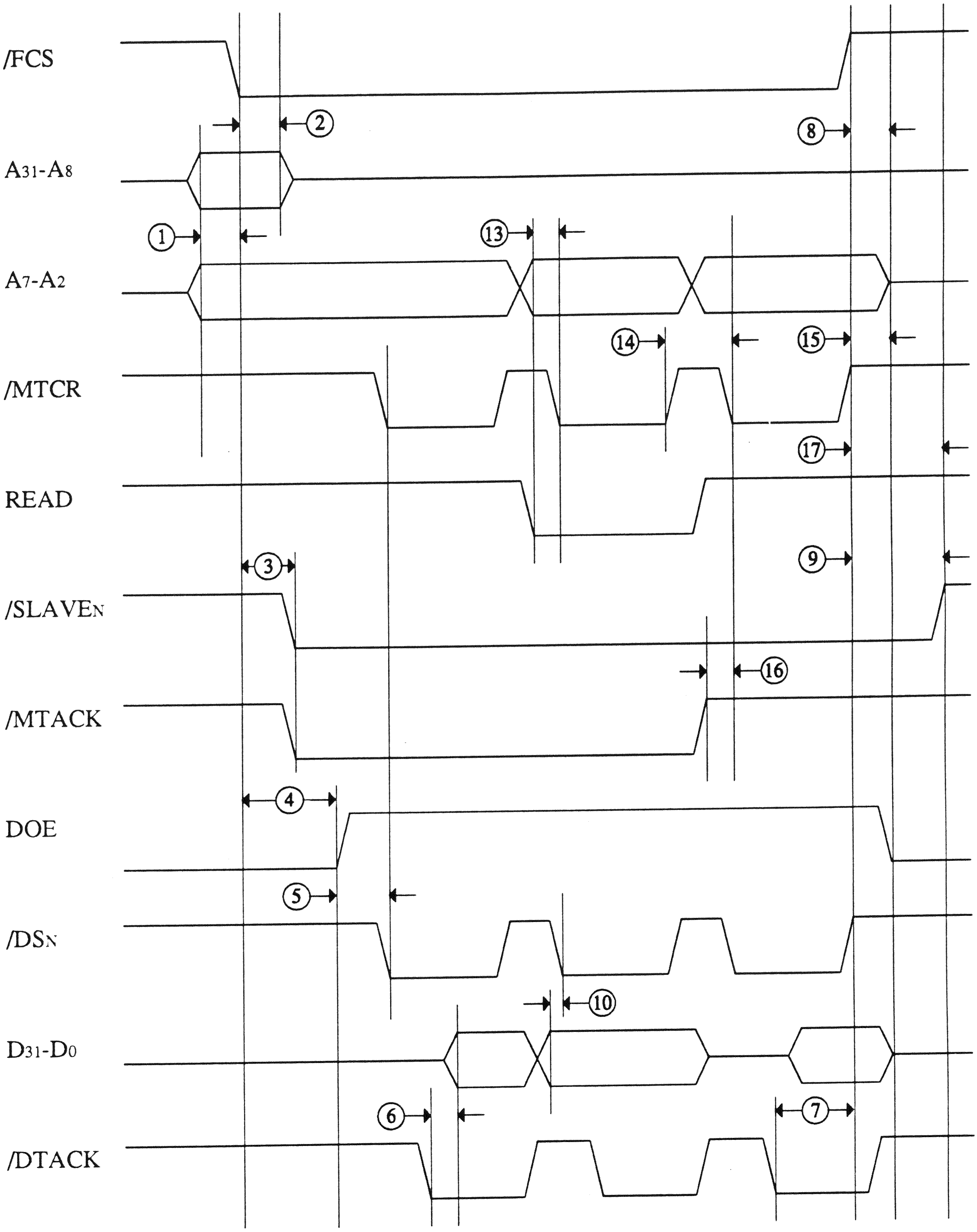
5.2 Standard Write Cycle Timing

No.	Name	Symbol	Min	Max
1	Address setup to /FCS	TAFS	15ns	-----
2	Address hold from /FCS	THAF	10ns	-----
3	/FCS to /SLAVEN delay	TSLV	-----	25ns
4	/FCS to DOE delay	TDOE	30ns	-----
5	DOE to /DSN delay	TDS	10ns	-----
7	/DTACK to /FCS off	TOFF	10ns	-----
8	Master signal hold from /FCS off	THMC	0ns	5ns
9	Slave signal hold from /FCS off	THSC	0ns	15ns
10	Write data setup to /DSN	TWDS	5ns	-----
11	/FCS to /CCS delay	TCCS	35ns	175ns
12	/CCS off to /FCS off	TOVL	40ns	-----


/FCS
A31-A8
A7-A2
READ
/SLAVEN
DOE
/DSN
D31-D0
/DTACK
/CCS
①
②
③
④
⑤
⑦
⑧
⑨
⑩
⑪
⑫

5.3 Multiple Transfer Cycle Timing

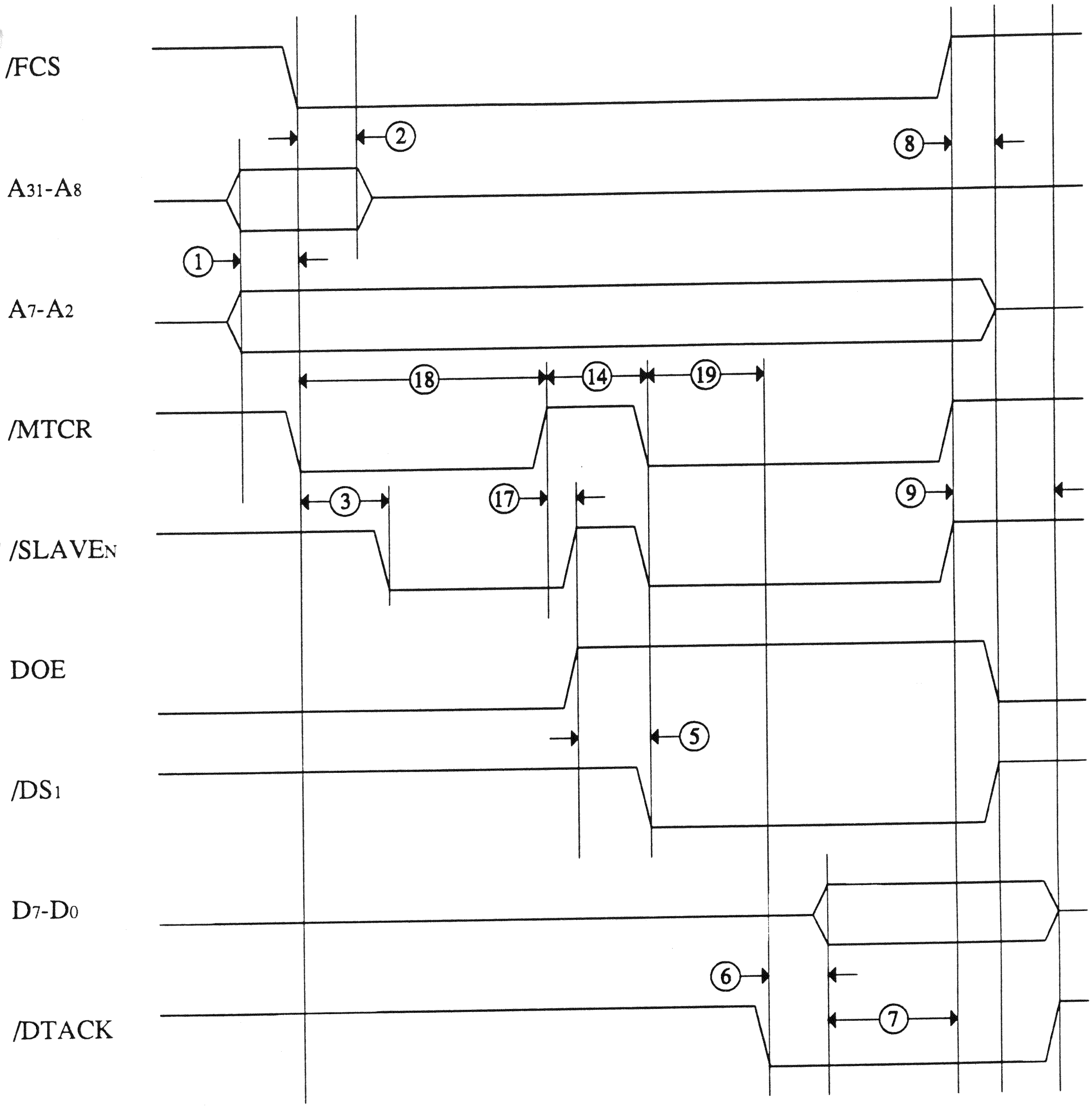
No.	Name	Symbol	Min	Max
1	Address setup to /FCS	T_{AFS}	15ns	-----
2	Address hold from /FCS	T_{HAF}	10ns	-----
3	/FCS to /SLAVE$_N$, /MTACK delay	T_{SLV}	-----	25ns
4	/FCS to DOE delay	T_{DOE}	30ns	-----
5	DOE to /DS$_N$, /MTCR delay	T_{DS}	10ns	-----
6	Data setup to /DTACK	T_{RDS}	0ns	-----
7	/DTACK to /FCS, /MTCR off	T_{OFF}	10ns	-----
8	Master signal hold from /FCS off	T_{HMC}	0ns	5ns
9	Slave signal hold from /FCS off	T_{HSC}	0ns	15ns
10	Write data setup to /DS$_N$	T_{WDS}	5ns	-----
13	Address, READ setup to /MTCR	T_{AMS}	5ns	-----
14	/MTCR off to /MTCR on	T_{REF}	10ns	-----
15	Address, READ hold from /MTCR	T_{HAM}	0ns	-----
16	/MTACK off to /MTCR	T_{BCD}	10ns	-----
17	Slave signal hold from /MTCR off	T_{HSM}	0ns	5ns



/FCS
A31-A8
A7-A2
/MTCR
READ
/SLAVEN
/MTACK
DOE
/DSN
D31-D0
/DTACK
①
②
③
④
⑤
⑥
⑦
⑧
⑨
⑩
⑬
⑭
⑮
⑯
⑰

5.4 Quick Interrupt Cycle Timing

No.	Name	Symbol	Min	Max
1	Address setup to /FCS	TAFS	15ns	-----
2	Address hold from /FCS	THAF	10ns	-----
3	/FCS to /SLAVEN delay	TSLV	-----	25ns
5	DOE to /DSN delay	TDS	10ns	-----
6	Data setup to /DTACK	TRDS	0ns	-----
7	/DTACK to /FCS off	TOFF	10ns	-----
8	Master signal hold from /FCS off	THMC	0ns	5ns
9	Slave signal hold from /FCS off	THSC	0ns	15ns
14	/MTCR off to /MTCR on	TREF	10ns	-----
17	Slave signal hold from /MTCR off	THSM	0ns	5ns
18	Poll Phase time	TPOL	30ns	100ns
19	Vector Phase start to /DTACK time	TVEC	-----	100ns


/FCS
A31-A8
A7-A2
/MTCR
/SLAVEN
DOE
/DS1
D7-D0
/DTACK
1
2
3
5
6
7
8
9
14
17
18
19

CHAPTER 6

ELECTRICAL SPECIFICATIONS

> *"...I collected the instruments of life around me, that I might infuse a spark of being into the lifeless thing that lay at my feet"*
>
> *-Victor Frankenstein*

The Zorro III bus has a number of electrical specifications that are very important for PIC designers to consider, along with the timing parameters of course. It's extremely important to base designs on the specification of the backplane, rather than the actual behavior of the backplane. New backplanes for new machines are designed to conform to the specification, they are not necessarily based on previous designs. This is expecially important with the Zorro III bus, since timing is far more critical than in the past, and the bus controller is designed from this specification, rather than the reverse, as in the Amiga 2000.

6.1 Expansion Bus Loading

The Zorro III bus loading is specified based on typical TTL family "F" series buffer devices, though in reality, compatible CMOS devices are likely to be used in some bus controllers or PICs. Thus, it's important to accept the TTL levels as a minimum voltage level, and make sure that all inputs are the appropriate TTL levels, while outputs can be at TTL or CMOS voltage levels as long as they provide the required source and sink.

While some A2000 designs used "LS" or "ALS" buffers instead of "F", the bus will generally work with these older cards, at least with current backplane designs such as the A3000 backplane. However, Zorro III designs must exactly obey these loading rules; it's very probable that some future Zorro III machines will have a large number of slots. In such machines, PICs built on the Zorro II specification will still work in a lightly loaded bus, but may not function in a fully loaded bus. All Zorro III PICs built to spec will work in any Zorro III backplane, without any loading problems, if all loading and timing rules are followed by the PIC designer. The bus signals are divided up into the four groups shown in Table 6-1, based on the loading characteristics of the particular signal. The signals in each group are given here.

Table 6-1: Zorro III Drive Types

Signal	Direction	High Level	Low Level
Standard	Loading Driven	+140µA @ +2.7VDC +2.5VDC @ -3.0mA	-3.2mA @ +0.4VDC +0.4VDC @ +64mA
Clock	Loading	+20µA @ +2.7VDC	-1.6mA @ +0.4VDC
O.C.	Loading Driven	+80µA @ +2.7VDC Not Driven	-3.2mA @ +0.4VDC +0.4VDC @ +20mA
Non-bussed	Loading Driven	+80µA @ +2.7VDC +2.5VDC @ -0.4mA	-1.0mA @ +0.4VDC +0.4VDC @ +4.0mA

6.1.1 Standard Signals

The majority of signals on the bus are in this group. These are bussed signals, driven actively on the bus by F-series (or compatible) drivers such as 74F245, usually tri-stated when ownership of the signal changed for master and slave, and generally terminated with a 220Ω/330Ω thevenin terminator. PICs can apply two standard loads to each of these signals when necessary.

/FCS
A_2-A_7
FC_0-FC_2
/MTCR
/CCS
AD_8-AD_{31}
DOE
/MTACK
$/DS_0$-$/DS_3$
SD_0-SD_7
/IORST
/LOCK
READ
/BCLR

6.1.2 Clock Signals

All clock signals on the bus are in this group. Many designs are very sensitive to clock delay, skew, and rise/fall times, so loading on the clock lines must be kept to a minimum. These are bussed signals, actively driven by the backplane, and source terminated with a low value series resistor. PICs can apply one standard load to each of these signals when necessary. Zorro II cards have the same clock rules, so there should never be clocking problems when using either card type in a backplane.

/C3
CDAC
/C1
7M
E Clock

6.1.3 Open Collector Signals

Many of the bus signals are shared via open collector or open drain outputs rather than via tri-stated signals; this is of course required for some asynchronous things like the shared interrupt lines, and it works well for other types of signals as well. Of course, a backplane resistor pulls these lines high, PICs only drive the line low.

/OWN
/BGACK
/CINH
/BERR
/DTACK
/RESET
/INT$_2$
/INT$_6$
/HLT

6.1.4 Non-bussed Signals

The non-bussed, or slot specific, signals are involved with only one slot on the bus (eg, each slot has its own copy). As a result, the drive requirements are much less for these signals. The backplane provides pullups or pulldowns, as required by the specific signal.

/CFGIN$_N$
/CFGOUT$_N$
/BR$_N$
/BG$_N$
SenseZ$_3$
/SLAVE$_N$

6.2 Slot Power Availability

The system power for the Zorro III bus is totally based on the slot configurations. A backplane is always free to supply extra power, but it must meet the minimum requirements specified here. All PICs must be designed with the minimum specifications in mind, especially the tolerances.

Pin	Supply
5,6	+5 VDC ± 5% @ 2 Amps
8	-5 VDC ± 5% @ 60 mA
10	+12 VDC ± 5% @ 500mA
20	-12 VDC ± 5% @ 60mA

6.3 Temperature Range

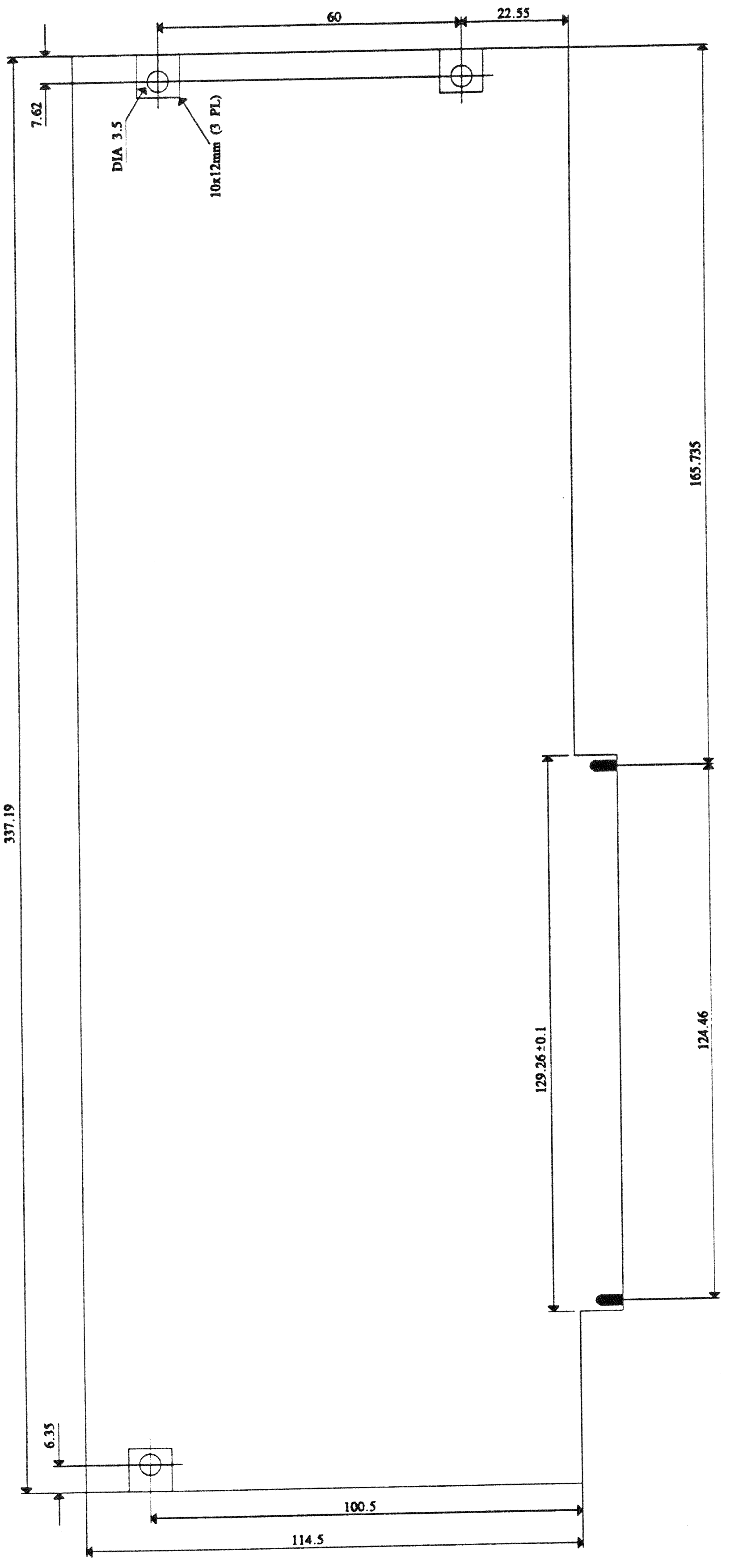
The Zorro III bus is specified for operation over a temperature range of 0° C to 70° C.

CHAPTER 7

MECHANICAL SPECIFICATIONS

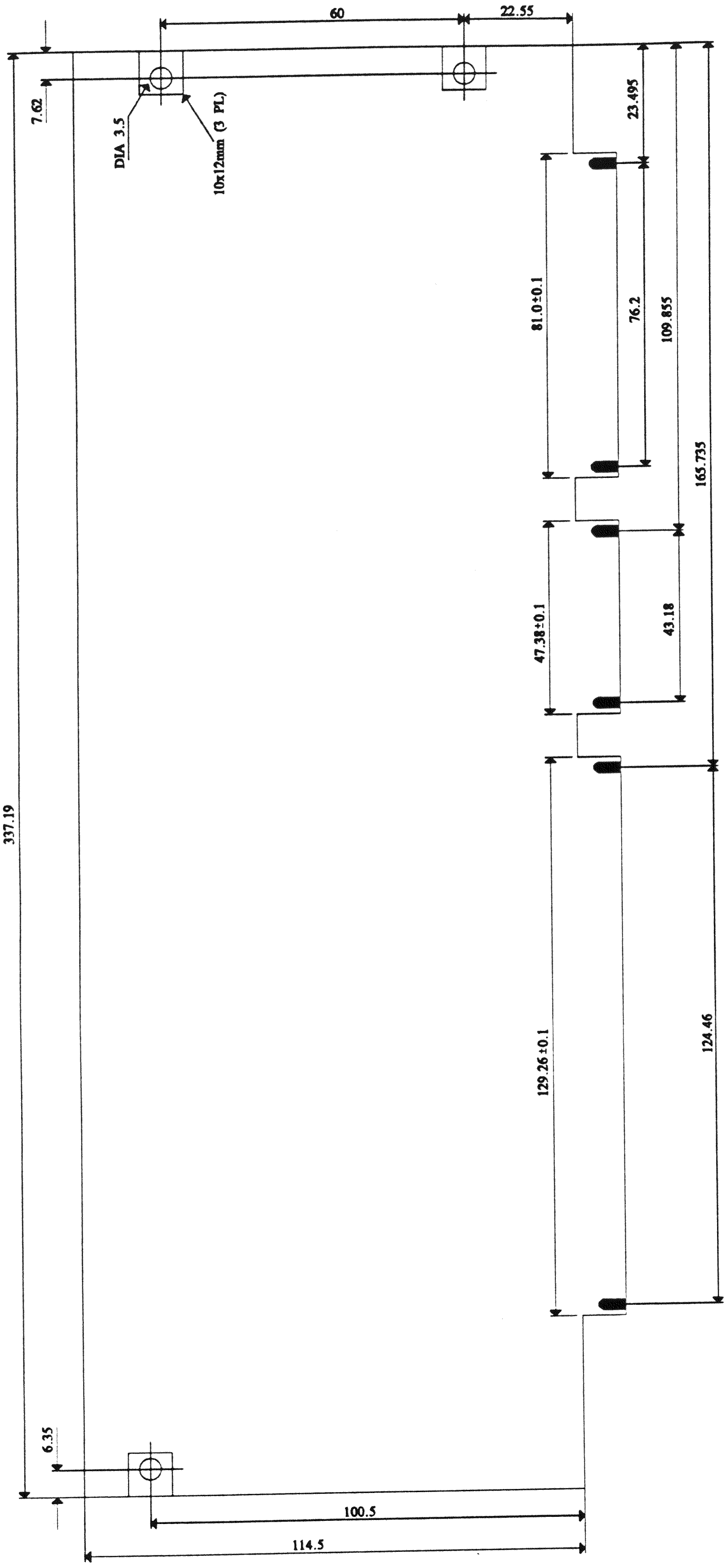
"Never speak more clearly than you think."
-Jeremy Bernstein

This section covers the various mechanical details of Zorro III cards. Note that these specifications are considered preliminary.


60
22.55
7.62
DIA 3.5
10x12mm (3 PL)
165.735
337.19
129.26±0.1
124.46
6.35
100.5
114.5

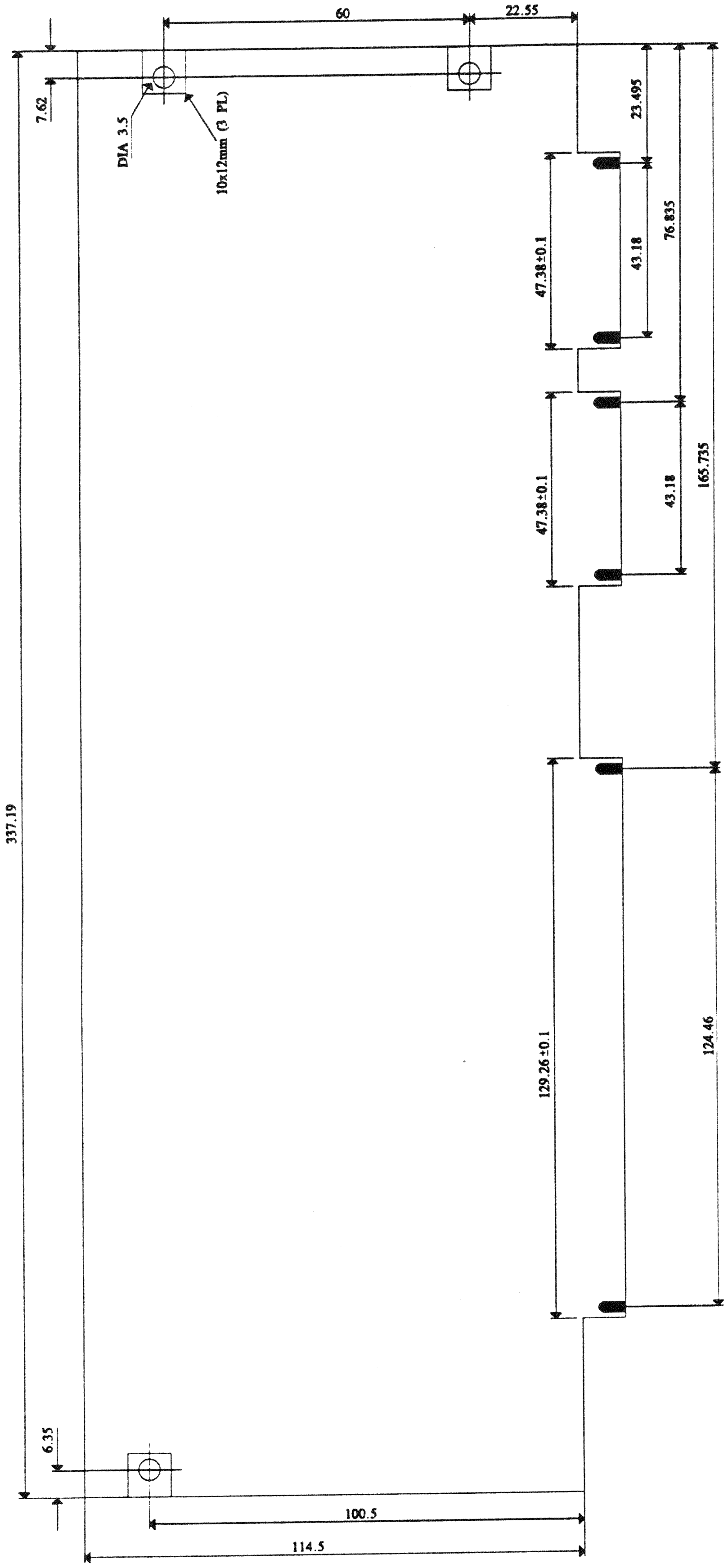
7.1 Basic Zorro III PIC

This drawing shows the basic Zorro III PIC. All of the dimensions are in millimeters.


60
22.55
7.62
DIA 3.5
10x12mm (3 PL)
23.495
81.0±0.1
76.2
109.855
165.735
47.38±0.1
43.18
337.19
129.26±0.1
124.46
6.35
100.5
114.5

7.2 PIC with ISA Option

This drawing shows the basic Zorro III PIC, with both Zorro III and the ISA Bus fingers specified. All of the dimensions are in millimeters.


60
22.55
7.62
DIA 3.5
10x12mm (3 PL)
23.495
47.38±0.1
43.18
76.835
47.38±0.1
43.18
165.735
337.19
129.26±0.1
124.46
6.35
100.5
114.5

7.3 PIC with Video Option

This drawing shows the basic Zorro III PIC, with both Zorro III and the Amiga Video Slot fingers specified. All of the dimensions are in millimeters. Please consult the *A500/A2000 Technical Reference Manual* for the form factor specification of a video-only card that will fit both Amiga 2000 and Amiga 3000 computers.

CHAPTER 8

AUTOCONFIG®

"The goal of all inanimate objects is to resist man and ultimately defeat him."
-Russell Baker

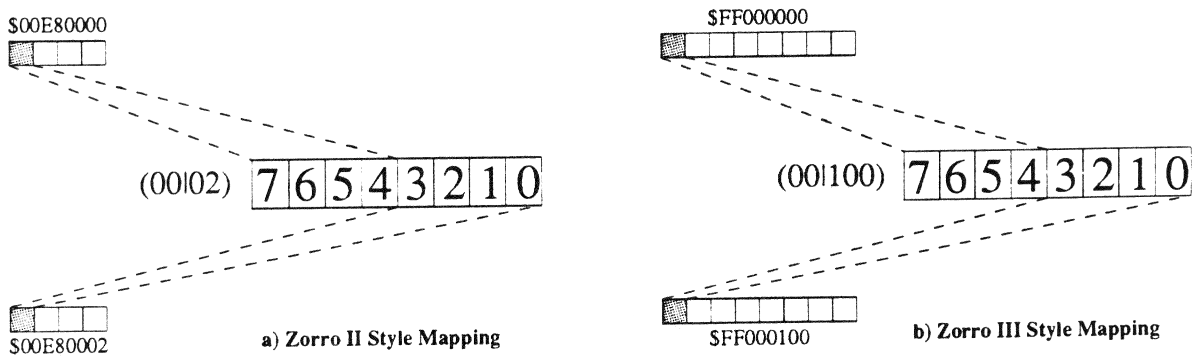
8.1 The AUTOCONFIG® Mechanism

The AUTOCONFIG® mechanism used for the Zorro III bus is an extension of the original Zorro II configuration mechanism. The main reason for this is that the Zorro II mechanism works so well, there was little need to change anything. The changes are simply support for new hardware features on the Zorro III bus.

Amiga autoconfiguration is surprisingly simple. When an Amiga powers up or resets, every card in the system goes to its unconfigured state. At this point, the most important signals in the system are /CFGINN and /CFGOUTN. As long as a card's /CFGINN line is negated, that card sits quietly and does nothing on the bus (though memory cards should continue to refresh even through reset, and any local board activities that don't concern the bus may take place after /RESET is negated). As part of the unconfigured state, /CFGOUTN is negated by the PIC immediately on reset.

The configuration process begins when a card's /CFGINN line is asserted, either by the backplane, if it's the first slot, or via the configuration chain, if it's a later card. The configuration chain simply ensures that only one unconfigured card will see an asserted /CFGINN at one time. An unconfigured card that sees its /CFGINN line asserted will respond to a block of memory called *configuration space*. In this block, the PIC will assert a set of read-only registers, followed by a set of write-only registers (the read-only registers are also known as AUTOCONFIG® ROM). Starting at the base of this block, the read registers describe the device's size, type, and other requirements. The operating system reads these, and based on them, decides what should be written to the board. Some write information is optional, but a board will always be assigned a base address or be told to shut up. The act of writing the final bit of base address, or writing anything to a shutup address, will cause the PIC to assert its /CFGOUTN, enabling the next board in the configuration chain.

The Zorro II configuration space is the 64K memory block $00E8xxxx, which of course is driven with 16 bit Zorro II cycles; all Zorro II cards configure there. The Zorro III configuration space is the 64K memory block beginning at $FF00xxxx, which is always driven with 32 bit Zorro III cycles (PICs need only decode A31-A24 during configuration). A Zorro III PIC can configure in Zorro II or Zorro III configuration space, at the designer's discretion, but not both at once. All read registers physically return only the top 4 bits of data, on D31-D28 for either bus mode. Write registers are written to support nybble, byte, and word registers for the same register, again based on what works best in hardware. This design attempts to map into real hardware as simply as possible. Every AUTOCONFIG® register is logically considered to be 8 bits wide; the 8 bits actually being nybbles from two paired addresses.



a) Zorro II Style Mapping

b) Zorro III Style Mapping

Figure 8-1: Configuration Register Mapping

The register mappings for the two different blocks are shown in *Figure 8-1.* All the bit patterns mentioned in the following sections are logical values. To avoid ambiguity, all registers are referred to by the number of the first register in the pair, since the first pair member is the same for both mapping schemes. In the actual implementation of these registers, all read registers except for the 00 register are physically complemented; eg, the logical value of register 3C is always 0, which means in hardware, the upper nybbles of locations $00E8003C and $00E8003E, or $FF00003C and $FF00013C, both return all 1's.

8.2 Register Bit Assignments

The actual register assignments are below. Most of the registers are the same as for the Zorro II bus, but are included here anyway for completeness. The Amiga OS software names for these registers in the ExpansionRom or ExpansionControl structures are included.

Reg Z2 Z3 Bit

00 02 100 7,6
(er_Type)

These bits encode the PIC type:

00	Reserved
01	Reserved
10	Zorro III
11	Zorro II

5 If this bit is set, the PIC's memory will be linked into the system free pool. The Zorro III register 08 may modify the size of the linked memory.

4 Setting this bit tells the OS to read an autoboot ROM.

3 This bit is set to indicate that the next board is related to this one; often logically separate PICs are physically located on the same card.

2-0 These bits indicate the configuration size of the PIC. This size can be modified for the Zorro III cards by the size extension bit, which is the new meaning of bit 5 in register 08.

Bits	Unextended	Extended
000	8 megabytes	16 megabytes
001	64 kilobytes	32 megabytes
010	128 kilobytes	64 megabytes
011	256 kilobytes	128 megabytes
100	512 kilobytes	256 megabytes
101	1 megabyte	512 megabytes
110	2 megabytes	1 gigabyte
111	4 megabytes	RESERVED

04 06 104 7-0
(er_Product)

The device's product number, which is completely up to the manufacturer. This is generally unique between different products, to help in identification of system cards, and it must be unique between devices using the automatic driver binding features.

08 0A 108 7
(er_Flags)

This was originally an indicator to place the card in the 8 megabyte Zorro II space, when set, or anywhere it'll fit, if cleared. Under the Zorro III spec, this is set to indicate that the board is basically a memory device, cleared to indicate that the board is basically an I/O device.

6 This bit is set to indicate that the board can't be shut up by software, cleared to indicate that the board can be shut up.

5 This is the size extension bit. If cleared, the size bits in register 00 mean the same as under Zorro II, if set, the size bits indicate a new size. The

most common new Zorro III sizes are the smaller ones; all new sized cards get aligned on their natural boundaries.

4 Reserved, must be 1 for all Zorro III cards.

3-0 These bits indicate a board's sub-size; the amount of memory actually required by a PIC. For memory boards that auto-link, this is the actual amount of memory that will be linked into the system free memory pool. A memory card, with memory starting at the base address, can be automatically sized by the Operating System. This sub-size option is intended to support cards with variable setups without requiring variable physical configuration capability on such cards. It also may greatly simplify a Zorro III design, since 16 megabyte cards and up can be designed with a single latch and comparator for base address matching, while 8 megabyte and smaller PICs require large latch/comparator circuits not available in standard TTL packages.

Bits	Encoding
0000	Logical size matches physical size
0001	Automatically sized by the Operating System
0010	64 kilobytes
0011	128 kilobytes
0100	256 kilobytes
0101	512 kilobytes
0110	1 megabyte
0111	2 megabytes
1000	4 megabytes
1001	6 megabytes
1010	8 megabytes
1011	10 megabytes
1100	12 megabytes
1101	14 megabytes
1110	Reserved
1111	Reserved

For boards that wish to be automatically sized by the operating system, a few rules apply. The memory is sized in 512K increments, and grows from the base address upward. Memory wraps are detected, but the design must insure that its data bus doesn't float when the sizing routine addresses memory locations that aren't physically present on the board; data bus pullups or pulldowns are recommended. This feature is designed to allow boards to be easily upgraded with additional or increased density memoried without the need for memory configuration jumpers.

Reg	Z2	Z3	Bit	
0C	0E	10C	7-0	Reserved, must be 0.
	(er_Reserved03)			
10	12	110	7-0	Manufacturer's number, high byte.
14	16	114	7-0	Manufacturer's number, low bytes. These are unique, and can only be assigned by Commodore.
	(er_Manufacturer)			
18	1A	118	7-0	Optional serial number, byte 0 (msb)
1C	1E	11C	7-0	Optional serial number, byte 1
20	22	120	7-0	Optional serial number, byte 2
24	26	124	7-0	Optional serial number, byte 3 (lsb)
	(er_SerialNumber)			This is for the manufacturer's use and can contain anything at all. The main intent is to allow a manufacturer to uniquely identify individual cards, but it can certainly be used for revision information or other data.
28	2A	128	7-0	Optional ROM vector, high byte.
2C	2E	12C	7-0	Optional ROM vector, low byte.
	(er_InitDiagVec)			If the ROM address valid bit (bit 4 of register (00|02)) is set, these two registers provide the sixteen bit offset from the board's base at which the start of the ROM code is located. If the ROM address valid bit is cleared, these registers are ignored.
30	32	130	7-0	Reserved, must be 0. Unsupported base register reset register under Zorro II*.
	(er_Reserved0c)			
34	36	134	7-0	Reserved, must be 0.
	(er_Reserved0d)			
38	3A	138	7-0	Reserved, must be 0.
	(er_Reserved0e)			
3C	3E	13C	7-0	Reserved, must be 0.
	(er_Reserved0f)			
40	42	140	7-0	Reserved, must be 0. Unsupported control state register under Zorro II*.
	(ec_Interrupt)			
44	46	144	7-0	High order base address register, write only.
48	4A	148	7-0	Low order base address register, write only.
	(ec_Z3_HighByte)			The high order register takes bits 31-24 of the board's configured address, the low ordered resgister takes bits 23-16. For Zorro III boards configured in the Zorro II space, the configuration address is written both nybble and byte wide, with the ordering:
	(ec_BaseAddress)			

---

*The original Zorro specifications called for a few registers, like these, that remained active after configuration. Support for this is impossible, since the configuration registers generally disappear when a board is configured, and absolutely must move out of the $00E8xxxx space. So since these couldn't really be implemented in hardware, system software has never supported them. They're included here for historical purposes.

Reg Z2 Z3 Bit

Reg	Nybble	Byte
46	A27-A24	N/A
44	A31-A28	A31-A24
4A	A19-A16	N/A
48	**A23-A20**	**A23-A16**

Note that writing to register 48 actually configures the board for both Zorro II and Zorro III boards in the Zorro II configuration block. For Zorro III PICs in the Zorro III configuration block, the action is slightly different. The software will actually write the configuration as byte and word wide accesses:

Reg	Byte	Word
48	A23-A16	N/A
44	**A31-A24**	**A31-A16**

The actual configuration takes place when register 44 is written, thus supporting any physical size of configuration register.

4C	4E	14C	7-0 (ec_Shutup)	Shut up register, write only. Anything written to 4C will cause a board that supports shut-up to completely disappear until the next reset.
50	52	150	7-0	Reserved, must be 0.
54	56	154	7-0	Reserved, must be 0.
58	5A	158	7-0	Reserved, must be 0.
5C	5E	15C	7-0	Reserved, must be 0.
60	62	160	7-0	Reserved, must be 0.
64	66	164	7-0	Reserved, must be 0.
68	6A	168	7-0	Reserved, must be 0.
6C	6E	16C	7-0	Reserved, must be 0.
70	72	170	7-0	Reserved, must be 0.
74	76	174	7-0	Reserved, must be 0.
78	7A	178	7-0	Reserved, must be 0.
7C	7E	17C	7-0	Reserved, must be 0.

APPENDICES

"I have been given the freedom to do as I see fit."
-REM

A.1 Physical and Logical Signal Names

The Amiga 3000 Bus signals vary based on the particular bus mode in effect. This table lists each physical pin by physical name, and then by the logical names for Zorro II mode, Zorro III mode, address phase, and Zorro III data mode, data phase.

PIN NO.	**Physical Name**	**Zorro II Name**	**Zorro III Address Phase**	**Zorro III Data Phase**
1	Ground	Ground	Ground	Ground
2	Ground	Ground	Ground	Ground
3	Ground	Ground	Ground	Ground
4	Ground	Ground	Ground	Ground
5	+5VDC	+5VDC	+5VDC	+5VDC
6	+5VDC	+5VDC	+5VDC	+5VDC
7	/OWN	/OWN	/OWN	/OWN
8	-5VDC	-5VDC	-5VDC	-5VDC
9	/SLAVE$_N$	/SLAVE$_N$	/SLAVE$_N$	/SLAVE$_N$
10	+12VDC	+12VDC	+12VDC	+12VDC
11	/CFGOUT$_N$	/CFGOUT$_N$	/CFGOUT$_N$	/CFGOUT$_N$
12	/CFGIN$_N$	/CFGIN$_N$	/CFGIN$_N$	/CFGIN$_N$
13	Ground	Ground	Ground	Ground
14	/C3	/C3 Clock	/C3 Clock	/C3 Clock
15	CDAC	CDAC Clock	CDAC Clock	CDAC Clock
16	/C1	/C1 Clock	/C1 Clock	/C1 Clock
17	/CINH	/OVR	/CINH	/CINH
18	/MTCR	XRDY	/MTCR	/MTCR
19	/INT$_2$	/INT$_2$	/INT$_2$	/INT$_2$
20	-12VDC	-12VDC	-12VDC	-12VDC
21	A_5	A_5	A_5	A_5
22	/INT$_6$	/INT$_6$	/INT$_6$	/INT$_6$
23	A_6	A_6	A_6	A_6
24	A_4	A_4	A_4	A_4
25	Ground	Ground	Ground	Ground
26	A_3	A_3	A_3	A_3
27	A_2	A_2	A_2	A_2
28	A_7	A_7	A_7	A_7
29	/LOCK	A_1	/LOCK	/LOCK
30	AD$_8$	A_8	A_8	D_0
31	FC$_0$	FC$_0$	FC$_0$	FC$_0$
32	AD$_9$	A_9	A_9	D_1
33	FC$_1$	FC$_1$	FC$_1$	FC$_1$
34	AD$_{10}$	A_{10}	A_{10}	D_2
35	FC$_2$	FC$_2$	FC$_2$	FC$_2$
36	AD$_{11}$	A_{11}	A_{11}	D_3
37	Ground	Ground	Ground	Ground
38	AD$_{12}$	A_{12}	A_{12}	D_4
39	AD$_{13}$	A_{13}	A_{13}	D_5
40	Reserved	(/EINT$_7$)	Reserved	Reserved
41	AD$_{14}$	A_{14}	A_{14}	D_6
42	Reserved	(/EINT$_5$)	Reserved	Reserved

PIN NO.	Physical Name	Zorro II Name	Zorro III Address Phase	Zorro III Data Phase
43	AD15	A15	A15	D7
44	Reserved	(/EINT4)	Reserved	Reserved
45	AD16	A16	A16	D8
46	/BERR	/BERR	/BERR	/BERR
47	AD17	A17	A17	D9
48	/MTACK	(/VPA)	/MTACK	/MTACK
49	Ground	Ground	Ground	Ground
50	E Clock	E Clock	E Clock	E Clock
51	/DS0	(/VMA)	/DS0	/DS0
52	AD18	A18	A18	D10
53	/RESET	/RST	/RESET	/RESET
54	AD19	A19	A19	D11
55	/HLT	/HLT	/HLT	/HLT
56	AD20	A20	A20	D12
57	AD22	A22	A22	D14
58	AD21	A21	A21	D13
59	AD23	A23	A23	D15
60	/BRN	/BRN	/BRN	/BRN
61	Ground	Ground	Ground	Ground
62	/BGACK	/BGACK	/BGACK	/BGACK
63	AD31	D15	A31	D31
64	/BGN	/BGN	/BGN	/BGN
65	AD30	D14	A30	D30
66	/DTACK	/DTACK	/DTACK	/DTACK
67	AD29	D13	A29	D29
68	READ	READ	READ	READ
69	AD28	D12	A28	D28
70	/DS2	/LDS	/DS2	/DS2
71	AD27	D11	A27	D27
72	/DS3	/UDS	/DS3	/DS3
73	Ground	Ground	Ground	Ground
74	/CCS	/AS	/CCS	/CCS
75	SD0	D0	Reserved	D16
76	AD26	D10	A26	D26
77	SD1	D1	Reserved	D17
78	AD25	D9	A25	D25
79	SD2	D2	Reserved	D18
80	AD24	D8	A24	D24
81	SD3	D3	Reserved	D19
82	SD7	D7	Reserved	D23
83	SD4	D4	Reserved	D20
84	SD6	D6	Reserved	D22

PIN NO.	Physical Name	Zorro II Name	Zorro III Address Phase	Zorro III Data Phase
85	Ground	Ground	Ground	Ground
86	SD_5	D_5	Reserved	D_{21}
87	Ground	Ground	Ground	Ground
88	Ground	Ground	Ground	Ground
89	Ground	Ground	Ground	Ground
90	Ground	Ground	Ground	Ground
91	$SenseZ_3$	Ground	$SenseZ_3$	$SenseZ_3$
92	7M	E7M	7M	7M
93	DOE	DOE	DOE	DOE
94	/IORST	/BUSRST	/IORST	/IORST
95	/BCLR	/GBG	/BCLR	/BCLR
96	Reserved	$(/EINT_1)$	Reserved	Reserved
97	/FCS	No Connect	/FCS	/FCS
98	$/DS_1$	No Connect	$/DS_1$	$/DS_1$
99	Ground	Ground	Ground	Ground
100	Ground	Ground	Ground	Ground

A.2 A Glossary of Terms

The reader may be unfamiliar with a number of terms used in this document. Every effort has been made to include all such terms here.

address	A byte-numbered memory location. The Zorro II bus is based on a 24 bit address, the Zorro III bus on a 32 bit address.
arbitration	The unambiguous selection of one request out of a number of possible simultaneous requests for a resource. There are two kinds of arbitration in a Zorro III system; bus arbitration and quick interrupt arbitration.
asserted	The active state of a state, regardless of its logic sense.
atomic cycle	A cycle or set of cycles that are uninterruptable, and thus treated as a unit; both Multiple Transfer and LOCKed cycles are considered atomic under the Zorro III bus.
AUTOCONFIG®	From "automatic configuration", the Zorro bus specification for how software and hardware cooperate to permit PIC addresses to be set by software and PIC type information to be determined by software. This is explained in Chapter 8, and in the *A500/A2000 Technical Reference Manual*, available from Commodore-Amiga.
backplane	The cage or motherboard subsection into which PICs are inserted. The Amiga 2000 and Amiga 3000 computers have integral backplanes, the Amiga 500 and Amiga 1000 computers require add-on backplane cages for Zorro II compatibility.
burst	A short name for Multiple Transfer Cycle mode. Essentially, within one full Zorro III cycle there can be any number of Multiple Transfer Cycles. Each full cycle has a complete 32 bit address supplied and a complete 32 bit datum transferred. Each burst cycle supplies only the 8 bit page address, but transfers a complete 32 bit datum faster than the standard full cycle would allow.
bus cycle	One complete bus transaction, indicated by the assertion of least one cycle strobe For any single bus cycle, there is one address, one data value, one data direction, and one cycle type in effect.
bus hogging	When a bus master takes over the bus for an undue amount of time. The Zorro II bus leaves it completely up to the individual PIC to avoid bus hogging; the Zorro III bus schedules PICs with the bus controller to evenly distribute the bus load.

bus starvation	When a master can't get access to the bus, it is said to be starved. On the Zorro II bus, two busy masters can completely starve a third master. Complete starvation is impossible on the Zorro III bus, though a bus hogging Zorro II card can cause similar symptoms.
byte	A collection of eight signals into a logical group, and the smallest independently addressable quantity on the Zorro bus.
clock	A free running signal driven at a fixed frequency to the bus, used mainly for clocking state machines on Zorro II cards.
cool	An unreachable goal for some, a way of life for others. The obvious example of this latter category being Dave Haynie, pictured at right.
cycle strobe	A bus signal that defines the boundary of a bus cycle; the Zorro II and Zorro II modes on a Zorro III bus each have their own cycle strobes. The current bus master always asserts the cycle strobes.
data	The contents of a memory location. The main purpose of a bus cycle is to transfer data between two locations. The Zorro II bus is based on a 16 bit data path, the Zorro III bus is based on a 32 bit data path.
DMA	Direct Memory Access; devices that have direct access to Zorro III slaves are said to have DMA capability. These devices are also called masters.
DMA latency	This is the time between a bus request and a bus grant as seen by a PIC wishing to become bus master.
device	A PIC, eg, a Zorro bus master or bus slave.
grant	The result of an arbitrated set of requests is a single grant; there are grants given for both the bus and quick interrupts.
Guinness	Attitude adjustment tonic, from Ireland. Said by some to be vital for sanity, if not normal human life.
hidden cycles	Cycles that occur on the local bus of a system, but can't be seen by devices on the expansion bus.
high	A signal driven to a logical +5V state is said to be high.
interrupt	An asynchronous line driven by a PIC to notify the CPU of some event, usually some hardware event governed by that PIC.

local bus	The main system bus of an Amiga computer is called the local bus. In general, the main CPU, video chips, chip memory, and any other built-in resources are on the local bus. The bus controller sits on both the local and expansion buses and manages the communications between them.
longword	Based on the Motorola conventions, a longword is equal to 4 bytes.
low	A signal driven to a logical +0V state is said to be low.
master	The device currently generating addresses for the expansion bus. There is only one master on the bus at a time, this being insured by the bus arbitration logic. The master also drives data on writes, the read, cycle, and data strobes, and several other signals.
motherboard	The main system circuit board for any Amiga computer. Resources on the local bus of a machine are often called motherboard resources.
negated	The inactive state of a signal, regardless of its logic sense.
nybble	A collection of four bits; one half of a byte. AUTOCONFIG® ROMs are physically nybble-wide.
paragraph	A sequence of closely related sentences, generally expressing and supporting one succinct idea. This term has no special computerese meaning in any rational, modern system.
PIC	Plug In Card. Any Amiga expansion card is called a PIC for short.
request	Asking for the use of some resource; the Zorro III bus has two kinds of requests, bus requests and quick interrupt requests.
slave	The device currently responding to the address on the expansion bus. There is only one slave on the bus at a time; an error is signalled by the bus collision detect logic if multiple slaves respond to the same address. The slave also drives data on reads, the transfer acknowledge strobe, and several other signals.
slot	A physical port on a Zorro backplane, which supplys independent /SLAVEN, /BRN, and /BGN lines, chained /CFGINN and /CFGOUTN lines, and is mechanically manifested as a 100 pin single-piece connector.
termination	Circuitry attached to a bus signal in order to minimize annoying analog things like ringing, reflections, crosstalk, and possibly random logic conditions which can arise when a bus is undriven.

timeout	A bus cycle terminated by the bus controller instead of by a responding slave device. If no slave responds to a bus cycle within a reasonable time period, the bus controller will terminate the cycle to prevent lockup of the system.
tri-state	A signal driven to a high impedence condition is said to be tri-stated.
word	Based on the Motorola conventions, a word is equal to 2 bytes.
Zorro	The name given to the Amiga bus specification. "Zorro I" refers to the original design for A1000 backplane boxes, "Zorro II" refers to the modification to this specification used for the A2000 and compatible backplanes, and "Zorro III" refers to the Zorro II compatible bus specification first used in the Amiga 3000 computer.

A.3 Zorro III Implementations

Functionally, there are two possible implementation levels in existance for the Zorro III bus. All of the features described in this focument are required for a full compliance Zorro III bus. However, the original Amiga 3000 computers were shipped with a bus controller that supported only a subset of the Zorro III specification published here. This is, however, upgradable.

The A3000 implementation of the Zorro III bus is driven by a custom controller chip called **Fat Buster**. The specification of this chip and the A3000 hardware are fully capable of supporting the complete Zorro III bus, but the initial silicon on Fat Buster, called the Level 1 Fat Buster, omits some features. Missing are:

- Support of Multiple Transfer Cycles.
- Support for Zorro III style bus arbitration.
- Support for Quick Interrupts.

The Level 2 version of Fat Buster has been in testing for some time at Commodore in West Chester, PA. Any developers who immediately intend to design PICs supporting these features are urged to contact Commodore Amiga Technical Support/Amiga Developer Support Europe for more information on obtaining samples of this part for use in A3000 systems. These parts are likely to be introduced into production, and available as part of an A3000 upgrade, very soon. All Buster chip revisions "13G" and earlier support the Level 1 features. Buster chip revisions "13H" and later support Level 2 features and improved Level 1 features as well.

BIGRAM 8/32

A Complete Zorro III PIC
Design Example

Document Revision 1.10

1991 DevCon Release

by Dave Haynie
July 18, 1991
Copyright © 1991 Commodore-Amiga, Inc.

IMPORTANT INFORMATION

"We don't know a millionth of one percent about anything."
-Thomas Alva Edison

This Document Contains Preliminary Information

The information contained here, while a honest attempt to illustrate a good Zorro III card design, is still preliminary in nature and subject to possible errors and omissions. We don't expect any problems with this design, but it's only responsible to supply you with this caveat.

Commodore Technology reserves the right to correct any mistake, error, omission, or viscious lie. Corrections will be published as updates to this document, which will be released as necessary in as developer-friendly a manner as possible. Revisions will be tracked via the revison number that appears on the front cover.

All information herein is Copyright © 1991 by Commodore-Amiga, Inc., and may not be reproduced in any form without permission.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 Intended Audience........ 1-1
1.2 A Few Words About AUTOCONFIG®........ 1-2
1.3 Design Example Goals........ 1-2

CHAPTER 2 AUTOCONFIG® LOGIC DESIGN

2.1 Bus Buffers........ 2-1
2.2 The AUTOCONFIG® ROM........ 2-2
2.3 The AUTOCONFIG® Registers........ 2-4
2.4 The SLAVE Logic........ 2-5

CHAPTER 3 MEMORY SYSTEM DESIGN

3.1 DRAM Refresh........ 3-1
3.1.1 Refresh Arbitration........ 3-2
3.1.2 Refresh Counter........ 3-2
3.1.3 Refresh Cycle........ 3-3
3.2 DRAM Access........ 3-4
3.2.1 Memory Cycle........ 3-5
3.2.2 Bank Selection........ 3-5
3.2.3 Address Multiplexing........ 3-6

CHAPTER 4 GOING FURTHER

4.1 Designed-In Enhancements........ 4-1
4.2 Modification Ideas........ 4-2

4.2.1 Tighter RAM Cycles.. 4-2
4.2.2 Read/Write Optimizations.. 4-3
4.2.3 Standard DRAM Tricks.. 4-3

CHAPTER 5 ADDITIONAL ZORRO III ADVICE

5.1 Watch Those Synchronizations!.. 5-1
5.2 Design for Speed.. 5-2
5.3 Watch Out For Noise!.............!.. 5-2
5.4 Follow the Specifications.. 5-3

APPENDICES

A.1 PAL Equations.. A-1
A.1.1 Autoconfiguration Control PAL.. A-2
A.1.2 Board Control PAL.. A-4
A.1.3 Memory Timing PAL.. A-6
A.1.4 CAS Control PAL.. A-8
A.1.5 Refresh Counter PAL.. A-10
A.2 Schematics.. A-12
A.3 Zorro III Configuration..
A-18

TABLES AND FIGURES

Table 2-1 Logical AUTOCONFIG® Registers.......... 2-2
Table 2-2 Physical ROM Registers.......... 2-3
Figure 3-1 Refresh Arbitration.......... 3-2
Figure 3-2 Refresh Cycle.......... 3-3
Figure 3-3 Memory Access.......... 3-4

CHAPTER 1

INTRODUCTION

"The curtain rises on a vast primitive wasteland,
not unlike certain parts of New Jersey."
-Woody Allen

This document fully describes an example Zorro III Plug-In-Card (PIC) design for a simple asynchronous dynamic RAM memory card. Its intent is to describe the procedures and underlying theories behind a basic Zorro III design. However, it is not a Zorro III designer's bible or any such rulebook. It should provide the designer with a better understanding of Zorro III PIC design, and perhaps provide a starting point for the beginning Amiga peripheral designer.

1.1 Intended Audience

This document was written primarily for hardware engineers interested in designing Plug-In-Cards for the Zorro III expansion bus. A reasonable level of microcomputer knowledge is a prerequisite to get much meaning out of these pages. A good understanding of the Zorro III bus theory, as outlined in *The Zorro III Bus Specification* (available from Commodore), is essential. Knowledge of basic TTL digital design with standard MSI and PAL devices is required, as is an understanding of dynamic RAMs. Familiarity with the Motorola 680x0 processors will also be quite useful.

While knowledge of Zorro II PIC design will also be useful, such experience mainly applies to the AUTOCONFIG® sections of a PIC design. The signals and design problems for the Zorro III bus are substantially different than for Zorro II. Zorro III PICs are expected to run considerably faster than those for Zorro II, leading the circuit designer to faster TTL logic families and more use of fast PAL devices. The additional speeds coupled with 32-bit buses will also lead the circuit board designer to multi-layer boards and more critical routing problems. While the Zorro II bus and most Zorro II designs are mainly synchronous, the Zorro III bus is asynchronous. Zorro III designs will typically be either fully asynchronous or self-clocked synchronous with proper attention to stable synchronization with the bus.

1.2 A Few Words About AUTOCONFIG®

If past history is any indication, the first thing to mention about Zorro III PIC design is AUTOCONFIG®, the Amiga mechanism for linking hardware plug-ins with software such that configuration jumpers for addresses are unnecessary, and device driver installation is trivial to even a novice user. And the first thing to say to a hardware designer about AUTOCONFIG® is *Don't Panic*. More than any other issue, the AUTOCONFIG® system seems to have confused Zorro II PIC designers. But there's absolutely nothing to fear about AUTOCONFIG®; it is a very simple concept and very simple to implement as an integral part of any PIC's design.

The concept of configuration hasn't changed for Zorro III, and the implementation is very much the same as for the Zorro II bus. Extensions have been provided for a few Zorro III advanced features, and a few extra things were added to the specification to make the design of a 32 bit PIC as easy as possible. Other than that, if you know Zorro II configuration, you'll pick up Zorro III configuration almost instantly. Chapter 2 walks through the creation of an AUTOCONFIG® circuit for Zorro III and discusses the basic logic likely to be in place on any Zorro III card.

1.3 Design Example Goals

The goal of this example is to design a memory card for the Zorro III bus. While A3000 users won't be running out of motherboard memory (up to 18 Megabytes) quite as fast as A2000 users did, there's already an emerging need for massive memory in Amiga computers. This RAM card meets the following goals:

- Provides a fully asynchronous design example
- Uses the same ZIP memories as the A3000
- Supports up to 8 Megabytes using 256K x 4 DRAMs, up to 32 Megabytes with 1M x 4 DRAMs.
- Hopefully functions as a realtively clear design example

And, of course, this is a fully functional design tested to the best of our ability at the time of this writing.

CHAPTER 2

AUTOCONFIG® LOGIC DESIGN

"Logic is in the eye of the logician."
-Gloria Steinem

Every PIC design has a few things in common, most noticably an AUTOCONFIG® circuit. While such logic can pretty much be created by rote, an optimal design always will incorporate the AUTOCONFIG® and other Zorro III bus logic naturally into the main design. While this chapter concentrates on the AUTOCONFIG® logic, it will cover all of the standard logic elements of any Zorro III design in a sensible order.

Throughout this and the following chapters, references to the schematic pages in Appendix 2 will be. Page one of the schematics is found on page A-13 of this document, and there are six schematic pages. To make things simpler, these will be referred to as S-1 through S-6.

2.1 Bus Buffers

Just like with Zorro II, all Zorro III designs require a number of buffers on the bus logic signals. No PIC may load any bus signal with more than two F-series equivalent gates, and of course outputs from the PIC must be able to drive the bus properly. Any unbuffered signal used by a PIC mut be used close to the bus connector; if a signal trace is longer than a few inches, it must be buffered. In addition, due to the dynamic nature of the high-order Zorro III address lines, some or all of these address lines must be latched for the duration of the bus cycle.

The buffering/latching arrangement is shown on S-1. Since this is a slave-only board, address lines are input-only. Addresses A_{31}-A_8 are transparently latched by 74F373 parts, the latch taking place when /FCS is negated. The transparent latching allows the address comparator to take advantage of the bus's address setup time, important for matching to the board's assigned address as quickly as possible. The circuitry shown here is the most straightforward, but in operation, only A_{24}-A_2 are actually used once the board select is determined. Thus, a fast enough comparator circuit can latch an address match rather than the high-order addresses if it saves on circuit complexity. Since the low order addresses A7-A2 are static, they are simply buffered coming into the RAM board. The extra buffers in that package are used in this design to buffer /FCS and READ, two lines used in several places in this design.

Data buffering is quite simple; D_{31}-D_0 are buffered with bidirectional bus buffers. The data direction and buffer enable signals are quite simple. The buffers point out toward the bus for read cycles when the PIC is selected (/SLAVE asserted), in at all other times; this function is contained in the U200 PAL. The output enable is asserted when the PIC is selected, the DOE signal is asserted, and there's no bus error; this function is contained in the U201 PAL. Because the data bus tristates, I use centering resistors to keep it quiet when it's not being driven. If this design had been supporting Zorro II as well as Zorro III, an additional two data buffers and much more complicated buffering logic, based on the SENSEZ3 line, would be required.

Reg	Bit	Val	Description
00	7,6	10	This indicates a Zorro III card.
	5	1	The OS will link this as free memory.
	4	0	No autoboot/diagnostic ROM.
	3	0	Only one logical PIC here.
	2-0	001	Using the extended size feature, this is a 32 megabyte board.
04	7-0	01010111	Commodore Product $53.
08	7	1	Hint to the OS that this is memory, not I/O
	6	0	This board can be shut up.
	5	1	Extended sizing being used here.
	4	1	This must be 1, for 1.3 compatibility.
	3-0	0001	Let the OS calculate the logical size of the memory.
0C	7-0	00000000	Reserved.
10	7-0	00000010	Manufacturer's number, high byte.
14	7-0	00000010	Manufacturer's number, low byte. Since this one is a Commodore board, it uses the Commodore number.
18-3C	7-0	00000000	All of these are zeroed. This board does not contain a board serial number or boot/diagnostic ROM.
40	7-0	N/A	Reserved
44	15-0	CFGADDR	This board uses the Zorro III configuration block. It accepts the configuration address as a single write.
48	7-0	N/A	Configuration is completely handled with register 44.
4C	7-0	N/A	A write of any value will cause the board to shut up.
50-7C	7-0	N/A	All remaining registers are reserved.

Table 2-1: Logical AUTOCONFIG® Registers

2.2 The AUTOCONFIG® ROM

The complete AUTOCONFIG® ROM is implemented in PAL U200, shown on schematic page S-2. The design of an AUTOCONFIG® ROM is usually very simple, but it does require a complete understanding of how the board is to be used by the system before it can be done. Also, a Zorro III configuration ROM is similar to a Zorro II configuration ROM, with just a few more options available, once the translation for the configuration space chosen is applied.

First of all, the board must be described. Obviously, this is a Zorro III memory board, and since it's my design, it's also from Commodore. On top of that, it can be expanded up to 32 megabytes, and it can also be "shut up" if necessary. That's pretty much the specification, now it has to be translated into Zorro III ROM registers. *The Zorro III Bus Specification* describes these entries starting on page 8-1. The logical register assignments are illustrated in *Table 2-1*. The table actually lists all of the configuration registers on the board (registers **40-7C** are reserved as write registers, not read registers, but they're mentioned here anyway).

The next step in the design process is to convert these bit assignments to actual logic. As mentioned before, the configuration ROM is implemented as part of the U200 PAL. By design, configuration ROMs fit nicely in a PAL in most cases. The Zorro II and Zorro III specifications call for all read resgisters other than register **00** to be inverted in their physical implementation. Since most bits are logically "0", they'll be physically "1", and "1" is the default output state of a standard PAL. Also taking into account that each logical register is actually made up of two physical registers, both of which assert data only on the D_{31}-D_{28} nybble, the physical register mapping for all read registers is shown in *Table 2-2*. The actual PAL equations for this are on page A-3. These are simply a set of equations, one for each data line, that take into account each "0" in the above table, and are active only when the board is selected and not yet configured.

Address	D_{31}	D_{30}	D_{29}	D_{28}
00	1	0	1	0
02	0	0	0	1
04	0	1	1	0
06	1	1	0	1
08	0	1	0	0
0A	1	1	1	0
12	1	1	0	1
16	1	1	0	1
OTHERS	1	1	1	1

Table 2-2: Physical ROM Registers

While it makes no difference to the equations for our ROM registers, it is a good idea to point out here the differences in addressing these read registers. Zorro II boards must respond to the configuration space $00E8xxxx, and all registers are mapped on word boundaries. Zorro III boards can respond to the $00E8xxxx address as a 16-bit Zorro II device as well, but many designs, including this one, will choose instead to respond to the Zorro III configuration space at $FF00xxxx. A board responds to this address as a 32-bit device, and it actually need only decode the high-order eight bits of this address; both of these facts can save considerably on the amount of configuration logic necessary for some designs. In both configurations, the first nybble of each register pair is at the offset from base address given by that register number. In the Zorro II space, the second nybble is in the next logical word -- the register number plus two. Zorro III instead maps the second register of the pair at $100 plus the register number. This may sound like the two will be quite different in implementation, but as the example PAL U200 illustrates, if I map A_8 as A_1 in the equations, all ROM equations will be written the same for either configuration space. Using this feature and a multiplex of A_8 and A_1 based on the SENSEZ3 signal can help simplify the design of a card that adjusts to both Zorro II and Zorro III buses.

2.3 The AUTOCONFIG® Registers

This design supports two writable configuration registers, the 16-bit configuration address register **44** and the shutup register **4C**. Recall that configuration address registers are written in a pattern that allows the designer to choose nybble- or byte-wide configuration latches for Zorro II configuration space or byte- or word-wide configuration latches in Zorro III configuration space. Since Zorro II space is only sixteen bits wide and writes must line up consistently, this design would have to latch configuration address bits A_{31}-A_{24} on a write to register **44**, followed by configuration address bits A_{23}-A_{16} on a write to register **48**. Even though a large board such as this never needs to look at A_{23}-A_{16} for its configuration address (Zorro III PICs always live at their natural boundaries), a board configured in Zorro II configuration space isn't configured until a write to register **48**. Since this board instead responds to Zorro III configuration space, the entire sixteen bit configuration address can be written at once with a write to register **44**, and that is also the signal indicating that configuration of the board is complete.

The register logic starts with the same PAL, U200, as used for our ROM logic. This PAL has the important low-order addresses going to it, so it's a natural for this. In this design, there are two signals created for register support in PAL U200. The first of these is a signal called /PRECON, for pre-configuration. The board isn't fully configured until the end of the Zorro III cycle that writes either register **44** or register **4C**; /PRECON is asserted during this last write cycle as soon as data is valid on the bus, and it stays latched until the next reset. The other signal in U200 that's of immediate importance is the CFGLT signal. This line is responsible for latching the configuration address on the bus if this final write is a configuration and not a "shut up" request. This is an active high signal in an inverted-output PAL, so the equation can't be very complicated. This line is asserted when the board is selected, /PRECON is asserted, and A_3 is low, which is true just after /PRECON is asserted for a write to **44**. Like the /PRECON line, CFGLT latches until the next reset. The remainder of the register logic is elsewhere.

The rest of the configuration control logic is in PAL U201, which creates both the /CFGOUT and /SLAVE signals, two signals that must be driven out to the backplane. The /CFGOUT signal is pretty simple. Normally, it is asserted at the end of a cycle in which /PRECON and /CFGIN are asserted, and latched asserted as long as PRECON also stays asserted. It also gets asserted if /CFGIN is asserted along with the SENSEZ3 signal negated. This latter condition indicates that the board has been placed in a Zorro II backplane. This board can't support Zorro II configuration, so it automatically "shuts up", an action required by the Zorro III specification. Note that the SENSEZ3 signal is called /Z2SHUNT in the PAL equations on page A-5.

The next basic piece of the configuration logic is the configuration latch, which in this case is the 74F374 at U202. This edge-triggered latch is triggered by the rising edge of CFGLT, which is asserted when the board's configuration address is written and data is valid on the data bus. At the end of the configuration address cycle, /CFGOUT is asserted, the address as latched is now fed into the /SLAVE generation address comparator, and the board is fully configured in hardware. Since this is an autosized memory board, system software generally will calculate its size and link it into the free memory pool before the next board is configured, though this operation can of course change as the configuration software changes.

2.4 The SLAVE Logic

Naturally, this brings up the question of how the /SLAVE logic is implemented. Every Zorro II or Zorro III board must assert its private /SLAVE line when it is responding to a bus address. In every case, two addresses must be supported; the configuration space address prior to configuration, and the software-assigned address after configuration. The method used in this example is quite similar to techniques used in many Zorro II designs, and is only slightly more complex.

The core of a /SLAVE circuit is always an address comparator of some kind. In every case, the bus address must be compared with the address to which the board responds. The main comparator in this circuit is the 74F521 at U203. It compares seven bits of possibly-latched bus address, A_{31}-A_{25}, with the corresponding bits on the configuration address latch. This comparison is called /MATCH on the schematics. Prior to configuration, the 74F374 is tri-stated, and the outputs going to the comparator are all pulled high, getting the card well on the way to responding to the $FF00xxxx configuration space.

The twist in this design is that there is a bit more to this comparison than just a simple comparator can handle. First of all, the board needs to look at a full eight bits of the $FF00xxxx address to properly respond during configuration, but only seven bits of address once the board is configured as a 32-megabyte board. This PAL U201 helps out by requiring A_{24} tobe high for a /SLAVE response prior to configuration. Zorro III memory cards must monitor the function codes FC_0-FC_2. PICs must only respond to a valid User or Supervisor mode Code or Data space access; such accesses are given as the exclusive-or of FC_0 with FC_1. The /SLAVE signal is always qualified with the Zorro III full cycle strobe /FCS, and it can occur in only two cases. In the first case, a qualified match occurs, the board is unconfigured, and /CFGIN is asserted. In the latter case, a qualified match occurs, the board is configured, and CFGLT is asserted. As previously mentioned, if the board is configured but CFGLT is negated, the board has been "shut up" rather than configured.

And that is all there is to the basic configuration logic. As demonstrated with U201, it is usually quite reasonable to incorporate this logic in with other board logic, where it'll fit the most efficiently. AUTOCONFIG® logic is intended to make it easy on the designer as well as the user; it's not supposed to scare anyone.

CHAPTER 3

MEMORY SYSTEM DESIGN

> *"I like them big and stupid."*
>
> *-Julie Brown*

This chapter discusses the actual DRAM logic design for this project. The information here is going to be far less useful for the designer trying to learn about Zorro III designs, since this is the part of the board design which is very specific to the task at hand, a DRAM board. However, there may be some ideas of a more general applicability. If nothing else, it shows that a fully asynchronous DRAM design can be done rather simply with a little planning.

3.1 DRAM Refresh

The most complex part of any hand-made DRAM circuitry is very likely to be the refresh circuitry. Without refresh, DRAM would look pretty much like static memory with a multiplexed address bus (and the folks at TI and National Semiconductor would be selling quite a few less DRAM controllers). While there's nothing wrong with off-the-shelf DRAM controllers, it's really not very difficult to "roll your own".

3.1.1 Refresh Arbitration

If you beilive that DRAM boards are difficult to design, and that refresh is the most difficult part of such a design, then you must believe that refresh arbitration is the most difficult part of the refresh logic. So I'll discuss that part first, and the rest of the circuitry in this chapter will then be simpler.

In this design the refresh arbiter is incorporated with the /SLAVE generator. Refresh arbitration takes place in U201 and is by the nature of the Zorro III bus necessarily asynchronous. A refresh request can come in at any time, and must be serviced as soon as possible without interrupting a cycle. There are three refresh cases: a request outside of a cycle, a request during a cycle to this card, and a request during a cycle to another card. These cases are illustrated in *Figure 3-1*. The problem with any asynchronous refresh arbiter is that it's impossible to determine at a single point if a cycle is starting or not. This can be though of as a potential race condition between the refresh request and the start-of-cycle. So the solution is to create two sampling points, one to give the go-ahead for a refresh cycle, the other to give the go-ahead for a memory cycle.

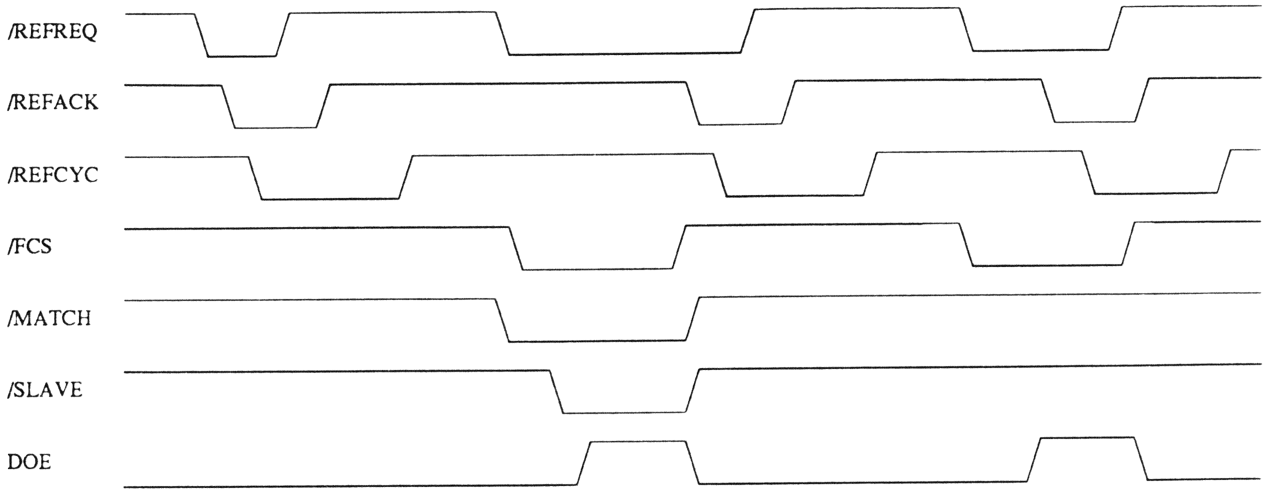


Figure 3-1: Refresh Arbitration

For the latter, you can use the /SLAVE signal. Virtually everything that happens on a simple Zorro III slave card is gated with /SLAVE. So in order to safely arbitrate refresh, we generate a refresh acknowledge signal, /REFACK, which will always be asserted safely before or safely after /SLAVE. In order to get there before /SLAVE, the /REFACK line will not be asserted outside of a Full Cycle if the /MATCH line is asserted. Since /MATCH and /FCS must both be asserted in order to create /SLAVE, and /FCS always follows /SLAVE, the /REFACK line is guaranteed to get out of U201 prior to /SLAVE, should the refresh request some in just before the PIC is selected. But once a board is selected on the bus, there's no reason to hold off refresh if it's a different board being selected, so /REFACK can be asserted during the data time of some other card's full bus cycle. In either case, the refresh acknowledge is latched as long as refresh request is held.

3.1.2 Refresh Counter

A simple refresh counter is implemented in PAL U306. Although the board supports both 256K x 4 or 1 Meg x 4 DRAM, the actual per-row refresh time is the same; the former part requires a 512 row refresh in 8ms, the latter a 1024 row refresh in 16ms. This amounts to one refresh request every 15,625ns. However, to build in support for burst mode with page-mode or static column DRAM, we use the TRAS,MAX time here, which is 10,000ns. The PAL counter actually counts 140ns clocks, so a count of 71 clocks will get us up to 9,940ns, close to the desired 10,000ns. If burst mode support weren't considered here, a count of 111 clocks could be used in the counter.

The counting is quite simple; the counter goes from zero to its terminal count, then asserts the /REFREQ signal. It then holds onto the /REFREQ signal until a refresh cycle is under way, as indicated by /REFCYC. The /REFCYC line will reset the counter for the duration of the refresh cycle. The process starts over once the refresh cycle is complete.

The clocked counter is used here simply because it's very easy to understand and, being fully digital, always works the same way. It could have been a simple one-shot or 555 timer circuit, as long as component tolerances don't allow the timer to drop below the required refresh frequency. You may recall reading of the evils of such timers in DRAM hint books. While they aren't optimal, due to the aforementioned component tolerance problems, that's not why you were warned off. The main reason for avoiding such timers in most DRAM designs is the problem you're likely to have with an asynchronous refresh request. Since we have already solved the problem of the asynchronous refresh request here, no asynchronous approach is inherently evil to this design.

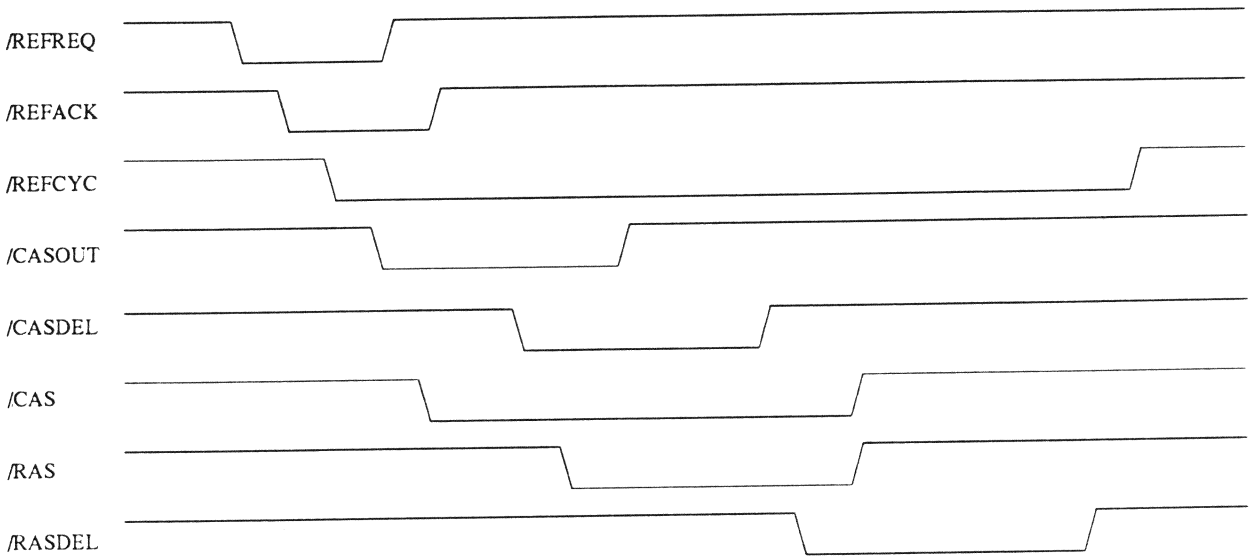


Figure 3-2: Refresh Cycle

3.1.3 Refresh Cycle

The actual refresh cycle, illustrated in *Figure 3-2*, is a CAS-before-RAS refresh, and all memory on board is refreshed at the same time. As soon as a refresh cycle is active (/REFCYC asserted), PAL U300 will assert the /REFCAS line. /REFCAS will in turn cause the CAS control PAL, U304, to drive all eight CAS lines. An active /CASDEL or /RASDEL will hold off the assertion of /REFCAS, thus ensuring RAS precharge (TRP) in case the refresh is immediately following a memory cycle. The /REFCAS line is latched by /MUX until /RASEN comes along, so that it's no longer dependent on /REFACK. The /REFACK line will be negated some time before the end of the CAS-RAS cycle; its main use here is to qualify the start of a refresh cycle. Once the /RASEN is asserted, /REFCAS is latched by the negated /RASDEL, as is /RASEN.

The /CASOUT line of U300 is also driven at the start of the refresh cycle. This of course comes back to U300 as the /CASDEL signal. The refresh /RASEN is driven as soon as /CASDEL is asserted, thereby separating refresh CAS and refresh RAS by roughly the CAS delay time. The /RASEN line drives the buffered /RAS lines to either bank of memory. Once asserted, /RASEN is held until /RASDEL wraps back in. The refresh cycle is held until /RASDEL once again is negated, thus ensuring TRP for the refresh cycle, in the event that this refresh is taking place right before a memory cycle.

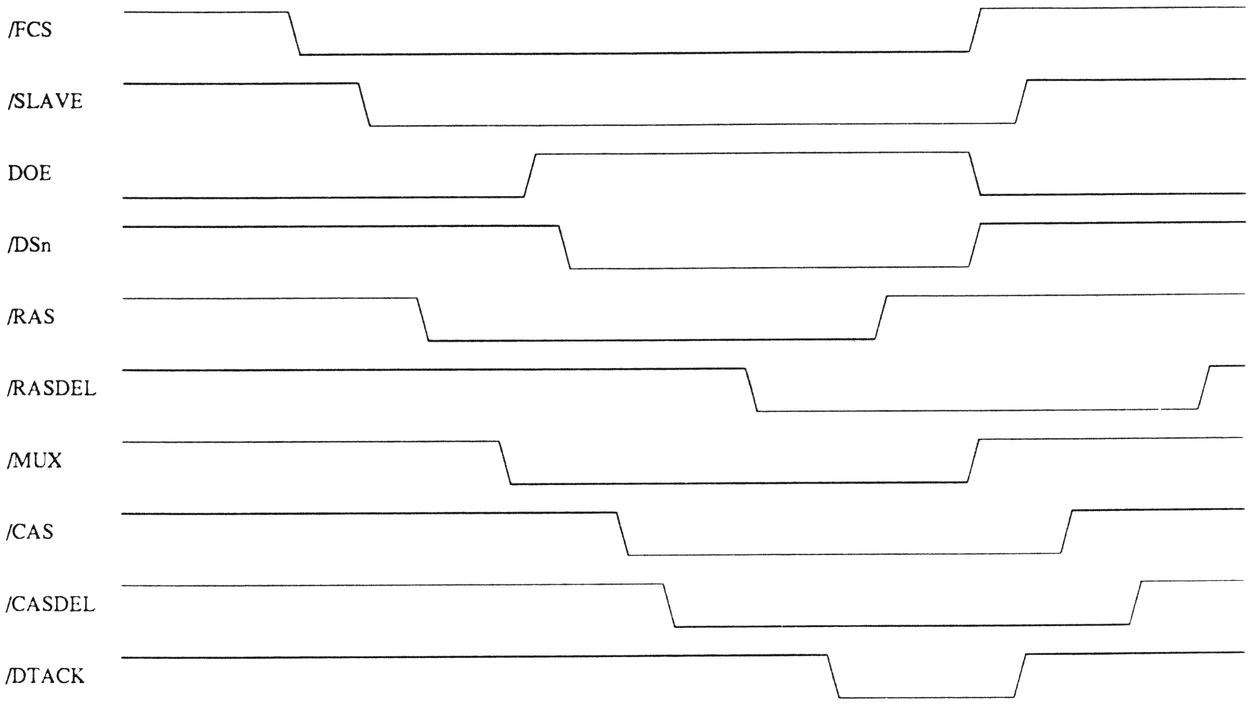


Figure 3-3: Memory Access

3.2 DRAM Access

The DRAM read/write access duing a normal memory cycle uses mainly the same parts for RAS-CAS controlling, along with a few additional bits and pieces to control memory banking. The logic supports several speeds of DRAM, selection being made via jumpers on the tap delays used for RAS and CAS timing. Either the 256K x 4 or 1 Meg x 4 parts can be used, and a jumper is provided to allow the necessary banking modification for this. Finally, Zorro III burst-mode (Multiple Transfer Cycle) support of either page or static column DRAMs is provided for. The design has been tested with a Level 2 Buster and fast-page DRAMs. While static column memories have not been tested, they should present no problem.

3.2.1 Memory Cycle

The basic memory cycle is started by U300 when /SLAVE is asserted and no refresh cycle is acknowledged or in progress. If burst is enabled, /MTCR is driven along with /SLAVE at the cycle's start. The memory cycle will be held off until /RASDEL is negated, to ensure T_{RP} after a refresh or a previous memory cycle. The assertion of /RASEN starts the cycle, and /RASEN is held through the end of the cycle. For fast-page memories, we could drop /RAS before cycle's end lets us gets an early start on RAS precharge. Since /RASEN creates /MUX, however, this optimization couldn't be used for SCRAM parts -- that could result in a column address change before cycle's end (SCRAMs don't latch the column address). Also, /RAS needs to be held through multiple /DTACKs in burst mode, so in this implementation the short /RASEN optimization is not done, though it's something to consider as an improvement. The 100ns tap delay U301 sets the RAS delay, and J300 provides taps for 100ns, 80ns, and 60ns DRAM. The /RASEN line is buffered, as previously mentioned, by two gates from the 74F244 at U303, one creating /RAS$_L$ for the lower bank of 32 memories, the other creating /RAS$_H$ for the upper bank of 32 memories. U303 also buffers the first tap from U301, which becomes /MUX, the line used for multiplexing the DRAM addresses.

The U300 PAL also creates the enable for CAS, the /CASEN line. This is based on /RASEN, DOE, and /MUX asserted, and it's held through the end of the cycle, until /DTACK is negated. The /CASEN line qualifies CAS, but it doesn't necessaily start CAS for a full cycle; further consideration of CAS generation is done elsewhere. For fast-page mode operation during a burst cycle, /CASEN follows /MTCR to generate a new CAS for each cycle. For static column operation during a burst cycle, /CASEN is simply held asserted until the cycle's end.

Most of the CAS generation is handled in U304, the CAS generation PAL. The CAS strobes are used to select between two banks of DRAM, and to select the appropriate bytes to access during write cycles; this is covered in detail in the next section. Other than qualifiying by bank and byte, the CAS generation PAL qualifies all CAS with READ. During read cycles, all four bytes in the accessed memory bank are activated, in order to support caching of this memory. Write cycles, on the other hand, are qualified with the appropriate data strobe, to assure that data is valid before a write-cycle CAS latches write data. All CAS strobes are of course qualified by /CASEN. They're also all qualified with /CADDR, which is a strobe that assures column address setup time to CAS. This is just the 60ns tap from the RAS timing tap delay. The 40ns tap would just about make it, but leaves absolutely no margin. Since column access is rarely the limiting factor, the 60ns tap is used, for a 30ns worst case /MUX to /CADDR delay, assuming a 5% per-tap tolerance on the tap delay.

3.2.2 Bank Selection

The refresh cycle's CAS-before-RAS logic, along with the fact that the whole board is refreshed at once, keeps things pretty simple when refresh is taking place. A normal memory cycle, however, must take into account the memory devices that actually need to be addressed. This discussion is concentrating mainly on the 256K x 4 devices, but the same principles apply to the 1M x 4 devices as well.

The basic memory unit is a 4-bit DRAM, and thus two devices are necessary to form a byte, the basic unit of interest to the Zorro III bus. This makes the smallest chunk of 32 bit memory a one megabyte chunk. So for the total of eight megabytes, we'll have eight 1-megabyte memory banks. We want to keep RAS common among all DRAM, so it can't be used to control banking at all. The best thing to do is divide and conquer, and that's just what we do; find something to select between these various natural divisions.

As mentioned previously, the /CAS strobes are used to select individual bytes within a one megabyte bank of memory. This is a very natural use of /CAS, since it's not needed until late in the memory cycle, and the data strobe lines and write data aren't valid until later in the cycle either. The CAS PAL could easily generate a /CAS$_0$-/CAS$_3$, based directly on corresponding data strobes /DS$_0$-/DS$_3$. However, there are twice the number of output lines on this PAL device as needed for four /CAS lines, and we're still looking for a banking mechanism. With the addition of the MEG4 signal for memory sizing and the address lines A_{22} and A_{24}, the PAL comes to drive eight total /CAS lines, controlling not only byte enables but the most significant RAM bank. For 256K x 4 parts, A_{22} chooses between two 4-megabyte banks. For 1M x 4 parts A_{24} chooses between two 16-megabyte banks.

Within the 4-megabyte banks, another banking control is used. In this case, most of the work is done by the 74F138 decoder at U305. This device creates a read enable for one of four device during a read, or a write enable for one of four devices during a write. The selection of device is controlled by the BK$_0$ and BK$_1$ lines from U300. BK$_0$ and BK$_1$ are simply A_{20} and A_{21} for 256K x 4 support, or A_{22} and A_{23} for 1M x 4 support. That's all there is to bank selection. Zorro III autosizing requires board memory to be added from the lowest to the highest address on-board, but there are no hardware requirements for this.

3.2.3 Address Multiplexing

There's nothing really complicated about the address multiplexing on this card, but it should be explained. All of the multiplexing is done with 74F258 multiplexers, and all of them are multiplexed by the /MUX signal. The first four or sixteen megabytes of memory is driven by /MAL$_0$-/MAL$_9$, the second by /MAH$_0$-MAH$_9$, but the multiplexing scheme is identical for both banks. When /MUX is high, the row addresses /MA$_0$-/MA$_9$ are set to the inverted A_{10}-A_{17}, A_{19}, and A_{21}, respectively. For /MUX low, the column addresses /MA$_0$-/MA$_9$ are set to the inverted A_2-A_9, A_{18}, and A_{20}, respectively. This organization may seem strange, but it makes A_2-A_7 (the Multiple Transfer static addresses), the low-order column addresses, so that Multiple Transfer Cycles can be supported via fast page or static column DRAM. This banking scheme also makes /MA$_9$, which is used only by 1M x 4 DRAM, a no-op for 256K x 4 DRAM, since BK$_0$-BK$_1$ look at A_{20} and A_{21}.

CHAPTER 4

GOING FURTHER

> *"There is more to life than increasing its speed."*
> *-Mahatma Gandhi*

In the general case, you can always do better. When specifics get involved, though, you may not always want to. In the specific case of this design example, you can certainly do a bit better. And if you want to make this example into a real product at some point, you should do better (thanks to this article, anyone can do at least this well just by copying).

Currently, with the Revision G Buster chip in a 25MHz A3000, this design with 80ns DRAM is running at just over half the speed of A3000 local bus memory. But part of that is the current Zorro III implementation -- this same configuration is running only about 15% slower than our prototype 50ns SRAM board! You can fully expect the Level 2 Buster chip to improve cycle times considerably, as well as supporting the faster MultipleTransfer Cycles. So, as I said, you can always do better.

4.1 Designed-In Enhancements

While not quite in the "quick and dirty" category, this example went from start to final working version in about five working days. Most of the careful design work was spent on getting the AUTOCONFIG® logic correct and understandable, since that's the most likely part of the design to be replicated in other Zorro III PICs. The actual DRAM part of it was designed, above all else, to work right the first time, since there really wasn't any time to revise the board. Because I felt that presenting a design example at a Developer's Conference without a working sample in hand would certainly be a cause for developers worry about the design's quality. So this card was designed to work, above all other concerns.

As it turns out, the original concept for the DRAM memory cycle worked fine, but the refresh logic has a rather serious flaw that hadn't been considered originally. When the design was created, the /REFACK signal was seen as the refresh control that stays valid for the entire refresh cycle, while the /REFCYC signal, then called /REFHOLD, was an end-of-cycle signal used to control the RAS precharge delay. That didn't work, and fortunately, the current mechanism could be created by changing the PAL equations, so the board was working a day after it was built up without a single cut or jumper.

However, the original memory cycle left a bit to be desired. Initially, the CAS enable didn't go out until the full RAS time had been met (eg, /RASDEL is asserted). This worked, but made CAS quite a bit later than it could have been. With a single extra wire, the CAS PAL was modified to hold off CAS until column addresses became valid. This allowed the memory timing PAL to enable CAS as soon as possible, and resulted in a 15% speedup.

The point here is that the design, as presented, isn't completely fixed. There are a considerable number of things one could do to change the memory cycle by playing around with PALs. It's conceivable that even without any additional PCB modifications, the memory cycle efficiency could be enhanced.

4.2 Modification Ideas

Opening up the design to a few PCB modifications can make things much more interesting. Of course, the ultimate modification might be to throw out the complete DRAM logic here and simply go to an off-the-shelf DRAM controller. While there's nothing wrong with that approach, and modern DRAM controllers even have an asynchronous operating mode that would work very nicely with Zorro III, there is still some performance that can be squeezed from this basic design. Most of these might have been incorporated with an extra day or so worth of design time.

4.2.1 Tighter RAM Cycles

The entire memory cycle run here is a bit less than optimal. Part of the problem is that the memory timing and CAS control PALs don't always have the same idea of when CAS should start. If the controller has a very good idea of when data is going to become valid, whether driven by TRAC or TCAS, the /DTACK line can be driven optimally. And, of course, the cycle can be fully TRAC driven, which is usually going to be the fastest possible cycle.

Another less than optimal feature of the design is the TRP assurance logic. In order to manage TRP between a cycle immediately following refresh or refresh immediately following a cycle, all new cycles are held off until /RASDEL is negated. This works just fine, but the time between /RAS negated and /RASDEL negated is very close to the TRAS time. For all standard DRAMs, the TRP time is less, sometimes much less, than the required time for TRAS. The CAS precharge time is never a problem for full cycle to full cycle operation, and unlikely to be a problem for Multiple Transfer Cycles.

The built-in support for Multiple Transfer Cycles can also be improved. The main problem for such burst cycles that doesn't crop up elsewhere is the TRAS,MAX time of most DRAMs in burst or static column modes. This board makes sure that a burst transfer can't exceed this limit by setting the refresh time to something just under TRAS,MAX. When refresh comes along, it causes /MTACK to be negated at the appropriate subcycle boundary, thus making the full cycle terminate so that refresh can take place. This has two shortcomings. First of all, it makes refresh related slowdowns over 50% more likely than necessary. Additionally, the start of the burst cycle isn't synchronized with the refresh counter, so a burst can be interrupted by refresh long before necessary. Ideally, separate counters could be added for burst and refresh timeouts. Alternately, the refresh counter could be modified to change its count based on whether or not a burst cycle is under way.

Additionally, this can cause bursts to last for strange counts. When a 68030 or 68040 driving the Zorro III bus it will ask for four count burst cycles. The 68030 can handle a shorter burst, but a 68040 can't. For that reason, this design will probably require that burst be disabled when used in an '040 based Amiga systems. While the Zorro III bus doesn't require it, it's a good idea to make sure that, if possible, designs that support burst will run at least four cycles. If the refresh counter, U306, were to hold off refresh requests during burst until at least four cycles had run, that would solve the problem.

4.2.2 Read/Write Optimizations

A basic principle of Zorro III slave optimization is that read and write cycles can benefit from different treatments. In this example, for instance, CAS can be driven before the DOE signal is received for read cycles, as long as column addresses are valid. If data can be valid on the card's data bus prior to DOE, then the cycle can be acknowledged only one buffer enable time after DOE is received. For READ sensing in the DRAM timing PAL (U300), the addresses used for the DRAM banking logic can easily be moved into another device, freeing up about seven pins.

Write optimizations would take a bit more logic, but they are possible. The best write enhancement would be data bus latches. By replacing 74F245 buffers U104-U107 with some 74F646 bidirectional latching buffers, and associated control logic, writes can be made very fast. The falling edge of the /DSN lines can latch data to the board and effect an immediate /DTACK, thereby possibly saving some of the TRAS and TCAS time. In fact, this could also help reads, since a latched data bus would allow the DRAMs to shut off as soon as data's latched, rather than at the end of the Zorro III cycle.

4.2.3 Standard DRAM Tricks

As with any DRAM design, the standard DRAM tricks apply here. With a bit of logic duplication, doubling up on the RAS-CAS and refresh logic, memory bank interleaving can be used to hide the RAS precharge time in most cases. Multiple Transfer Cycles can be though of as an automatic page detect, so conventional page mode or static column optimizations may not be all that useful. Then again, the Zorro III page is only 256 bytes, so perhaps a larger page could be of some help. Nybble mode memories won't really be of much use; although any burst cycle resulting from 68030 burst mode will be nybble compatible, there's no guarantee of linear addressing within a Zorro III burst cycle.

Always keep in mind the future. The Zorro III bus implementation that's currently on the A3000, as mentioned before, is already slated to improve. In the future memory will go faster on the bus than it does now, even if motherboard clocks don't go beyond 25MHz. And we expect future Zorro III machines will be running a faster Zorro III bus, going beyond what's possible in an A3000 even tomorrow.

CHAPTER 5

ADDITIONAL ZORRO III ADVICE

"Cute rots the intellect."

-Garfield

Going beyond this specific example just a bit, there are a few good things to think about when working on any Zorro III design. A large portion of this is just plain good design sense. Those without much design sense or experience should read this chapter twice, and probably learn more about the first two points from some outside materials.

5.1 Watch Those Synchronizations!

The foremost thing to be concerned about when designing for Zorro III is the fact the the bus is running asynchronously. Fully asynchronous designs, of course, will not find this to be any problem. Obviously a simply I/O chip with a 100ns access time can be timed with a delay line, keeping things very simple. Self-timed devices work real well, and are easy to hook into a Zorro III design. At the other end of the spectrum of complexity, clever clocked VLSI chips often internally synchronize things, much the way the 680x0 processors handle their "asynchronous" inputs, appearing to the world as if they're capable of running asynchronously.

If, however, you're doing your own TTL level design, such as this one, be very careful. Fully asynchronous circuits can be very tricky to design correctly; missing a strobe by a nanosecond or so can be fatal, and it may only happen every so often. The best bet is to use overlapping signals and feedback to create new signals, and *never* count on delays through PALs or TTL to provide repeatable delays. Tap delays, while not perfect, are reasonably accurate, and can be used to design reliable circuits. Inherently synchronous devices, such as many modern microprocessors, must be synchronized with external logic.

Synchronous design is usually easier, and therefore more reliable for the average designer to create. The problem here is coupling the synchronous design to the Zorro III bus. Such a design will have its own clock, but that clock can't reliably sample any Zorro III signal on a single edge. Double clocking any important Zorro III inputs with high quality flip-flops that go to clocked logic is a necessity. The problem you'll have with single clocking, metastability, won't always be immediately noticed, but it's going to be there. Better to avoid it from the start.

5.2 Design for Speed

Zorro III cards currently run around four times faster than Zorro II cards, and the limit, at least in theory, is over ten times faster. That should be a good indication that Zorro III designs are more sensitive to problems than Zorro II cards. To further aggravate the situation, you may not see any problems until faster Zorro III bus masters come along. So proper design practices are your best option. There are three main design problems that typically come up.

The speed of the design is one problem area, though it's not that much of a problem if you're up-to-date on the logic of the 1990s. While FCT and F series TTL are good for buffers and small logic functions, most fast designs these days rely heavily on programmable logic, mainly PALs. A single level of PAL logic can replace several levels of TTL, and they're aways pushing PAL speeds just a little bit more. Larger PLDs and gate arrays (programmable or custom) are always handy for complex circuits, providing they're fast enough.

5.3 Watch Out For Noise!

Noise problems are partially a result of the higher speeds involved. Eliminating such problems is achieved via a combination of circuit design and PCB layout. For noise reducing design, you need bypass capacitors of various sizes in the appropriate places. Every TTL part should have a small capacitor; we generally use something in the 0.1µF-0.22µF range. For DRAM or other surging parts, we use 0.33µF or greater. It's also a good idea to have a high frequency bypass, maybe 0.01µF or so, and a couple of larger capacitors, something in the 10µF-100µF range, randomly distributed around the design. More noise reduction can be achieved with good signal termination. Small value series termination resistors, something in the 22Ω-68Ω range works well; the values must often be tuned to the design. Tri-statable buses often benefit from some kind of parallel termination; pullups, pulldowns, or centering resistors depending on the design.

The other half of the noise problem is solved in PCB layout. Zorro III boards are expected to be multi-layer boards. Trace lengths are to be kept as short as possible, especially those on the bus side of a card; it's extremely important to minimize the noise that a card introduces to the bus. Fast and noisy signals, such as clock lines or fast control signals, should generally be given priority when routed. Component placement is also a very important job; the lengths of interconnects is directly affected by this planning. If the circuit designer isn't doing the board layout personally, he/she should develop a good working relationship with the PCB designer. If your CAD system allows the designer to specify signal importance for layout, use that feature. Any work done on keeping the design quiet will very likely be time well spent; it's going to help out in reliability, operation with other boards in the system, and government noise certifications such as FCC or FTZ.

5.4 Follow the Specifications

Let's say it once again! Any current Zorro III bus implementation is likely to be far more relaxed than the bus specification. That's going to eventually change. A proper design built today should work in tomorrow's 50MHz superAmiga, a substandard design could fail on an A3000 with the Buster Level 2 chip (Level 2 Buster drives the Zorro III bus approximately 10% faster then Level 1 does). Build in your long term viability at the design stage and save a great deal of potential future grief. You may not get fully tested on your design for some time to come.

APPENDICES

"It ain't the meat, it's the motion"
-Southside Johnny

A.1 PAL Equations

The following section contains the complete PAL equations for the five PAL devices in the BIGRAM design. All the equations are in the CUPL™ format, but should be easily translated to any other format if required. This format uses the & character to represent AND, the # symbol to represent OR, the $ symbol for XOR, and the ! symbol for negation. Standard outputs are indicated simply by name, registered outputs are indicated with the .D extension, and output enables are indicated with the .OE extension. The CUPL™ compiler minimizes equations where possible; should any equations here appear to be too large, rest assured that they will actually fit in the specified PAL.

A.1.1 Autoconfiguration Control PAL

This device is responsible for providing the AUTOCONFIG® ROM, registers, and data buffer direction control. This is to be programmed into a 15ns 16L8 or equivalent device.

```
            PARTNO      U200 ;
            NAME        U200 ;
            DATE        May 30, 1990 ;
            REV         2;
            DESIGNER    Dave Haynie ;
            COMPANY     Commodore-Amiga ;
            ASSEMBLY    BIGRAM ;
            LOCATION    U200 ;

/******************************************************************/
/*                                                                */
/* Zorro III BIGRAM Configuration Control                         */
/*                                                                */
/*      This device acts as configuration ROM and configuration   */
/*      register controller.                                      */
/*                                                                */
/******************************************************************/
/*                                                                */
/* DEVICE DATA:                                                   */
/*                                                                */
/*      Device:     16L8-15                                       */
/*      Clock:      NONE                                          */
/*      Unused:     NONE                                          */
/*                                                                */
/******************************************************************/

/* INPUTS: */

PIN 1   =           !SLAVE      ;       /* Board selected? */
PIN 2   =           !RST        ;       /* Board reset */
PIN 3   =           !DS3        ;       /* High order data strobe. */
PIN 4   =            READ       ;       /* Read cycle strobe */
PIN 5   =            A2         ;       /* Bus Addresses. */
PIN 6   =            A3         ;
PIN 7   =            A4         ;
PIN 8   =            A5         ;
PIN 9   =            A6         ;
PIN 11  =            A1         ;       /* This is really A8. */
PIN 16  =           !CFGOUT     ;       /* Board configured? */

/* OUTPUTS: */

PIN 19  =            D28        ;       /* Configuration data ROM nybble. */
PIN 12  =            D31        ;
PIN 13  =            D30        ;
PIN 14  =            D29        ;
PIN 15  =            DBDIR      ;       /* Data buffer direction. */

/* BIDIRECTIONALS: */

PIN 17  =           !PRECON     ;       /* Preconfiguation strobe. */
PIN 18  =            CFGLT      ;       /* Configuration address latch. */


/** INTERNAL TERMS: **/

/* Mapping A8 as A1 here makes the register pairs line up just
as they would under Zorro II configuration. */

field addr          = [A6..1];


/** OUTPUT TERMS: **/

/* The configuration ROM is created here.  The logical ordering
of it is as follows:

        REG         76543210

        00          10100001    Zorro III, autolink, 32 megabytes
        04          10010010    Product $53
        08          10110001    Extended Memory board, supports
                                 Shutup, autosized in software.
        0C          00000000    Reserved
        10          00000010    Manufacturer's code (C-A)
        14          00000010
        18-3C       00000000    Zeroed options/reserved.

The autoconfiguration specs call for every readable register
except for 0 to be inverted in the physical implementation.
So the resulting map is:
        ADDR        D31 D30 D29 D28
         00          1   0   1   0
         02          0   0   0   1
         04          0   1   1   0
         06          1   1   0   1
         08          0   1   0   0
         0A          1   1   1   0
         0C          1   1   1   1
         0E          1   1   1   1
         10          1   1   1   1
         12          1   1   0   1
         14          1   1   1   1
         16          1   1   0   1
        OTHERS       1   1   1   1

Only the Zero terms are explicitly entered here; anything not specifically
driven low will be driven high.
*/

!D31                = addr:02
                    # addr:04
                    # addr:08;

!D30                = addr:00
                    # addr:02;

!D29                = addr:02
                    # addr:06
                    # addr:08
                    # addr:12
                    # addr:16;

!D28                = addr:00
                    # addr:04
                    # addr:08
                    # addr:0A;

[D31..28].OE        = SLAVE & !CFGOUT & READ;

/* This signal is driven to indicate an address latch request.
Note that the board uses 16 bit configuration write feature
to configure all at once; this isn't available in the Zorro II
configuration space. */

CFGLT               = SLAVE  & PRECON & !A3
                    # CFGLT  & !RST;

/* If the board is told to shut up or configure, this line is
asserted and held through reset.  The logical SHUTUP line
is PRECON & !CFGLT, once FCS is negated. */

PRECON              = SLAVE & DS3 & !READ  &  addr:4C
                    # SLAVE & DS3 & !READ & addr:44
                    # PRECON & !RST;

/* This controls the data buffer direction between the PIC's
local bus and the expansion bus. */

DBDIR               = SLAVE & READ;
```

A.1.2 Board Control PAL

This device controls an assortment of board functions. It creates the /SLAVE, /CFGOUT, and /MTACK signals for Zorro III. It creates the data buffer enable for the bus buffers, and the burst-enable line used by the memory system. And it arbitrates DRAM refresh. This is programmed into a 10ns 20L8 or equivalent PAL.

```
          PARTNO      U201 ;
          NAME        U201 ;
          DATE        May 30, 1990 ;
          REV         3 ;
          DESIGNER    Dave Haynie ;
          COMPANY     Commodore-Amiga ;
          ASSEMBLY    BIGRAM ;
          LOCATION    U201 ;

/***************************************************************************/
/*                                                                         */
/* Zorro III BIGRAM Board Control                                          */
/*                                                                         */
/*      This device controls the main features of the BIGRAM board.        */
/*                                                                         */
/***************************************************************************/
/*                                                                         */
/* DEVICE DATA:                                                            */
/*                                                                         */
/*      Device:     20L8-10                                                */
/*      Clock:      NONE                                                   */
/*      Unused:     22(O)                                                  */
/*                                                                         */
/***************************************************************************/

/* INPUTS: */

PIN 1   =           !MATCH      ;       /* Address match from comparator. */
PIN 2   =            CFGLT      ;       /* Configuration latch. */
PIN 3   =           !PRECON     ;       /* Board was configed or shutup. */
PIN 4   =           !FCS        ;       /* Full Cycle Strobe. */
PIN 5   =           !CFGIN      ;       /* Configuration chain in. */
PIN 6   =            FC0        ;       /* Function codes, don't ignore these! */
PIN 7   =            FC1        ;
PIN 8   =           !REFREQ     ;       /* Refresh request from refresh counter */
PIN 9   =           !Z2SHUNT    ;       /* Zorro II backplane bypass. */
PIN 10  =            DOE        ;       /* Data enable. */
PIN 11  =           !BERR       ;       /* Bus error, all off. */
PIN 13  =           !REFCYC     ;       /* We're in a refresh cycle. */
PIN 14  =           !BRENB      ;       /* Burst/Multiple transfer enable. */
PIN 20  =           !MTCR       ;       /* We're in a multiple cycle. */
PIN 23  =            A24        ;       /* Latched bus address 24. */

/* OUTPUTS: */

PIN 15  =           !DBOE       ;       /* Data buffer output enable. */

/* BIDIRECTIONALS: */

PIN 16  =           !CFGOUT     ;       /* Board is configured. */
PIN 17  =           !REFACK     ;       /* Refresh acknowledge. */
PIN 18  =           !MTACK      ;       /* Multiple transfer acknowledge. */
PIN 19  =           !SLAVE      ;       /* Board select. */
PIN 21  =           !BURST      ;       /* This is a burst cycle. */

/** INTERNAL TERMS: **/

/* The valid board address consists of a comparator match and a valid
memory space.  The valid spaces are as follows:

        SPACE                   FC2 FC1 FC0
        Reserved                 0   0   0
        User Data                0   0   1
        User Program             0   1   0
        Reserved                 0   1   1
        Reserved                 1   0   0
        Supervisor Data          1   0   1
        Supervisor Program       1   1   0
        CPU                      1   1   1

This reduces to the equation used: FC0 XOR FC1. The external comparator
only looks at A31..A25, which is OK for normal operation (we're a 32
meg board), but bad for configuration.  So if we're not yet configured,
A24 must be high for a select match.
*/

select              = MATCH & (FC0 $ FC1) & (CFGOUT # A24);
```

```
/* This indicates a normal board select; SLAVE starts the cycle, FCS
cuts it off quickly at the end. */

hit                 = SLAVE & FCS;

/* OUTPUT TERMS: */

/* This output controls the data buffer enable pins.  Data buffers
turn on when DOE is asserted and the board is selected, they
turn off as quickly after a cycle ends as possible. */

DBOE                = hit & DOE & !BERR;

/* This signal indicates that the board is configured. The board is
considered configured if actually configured, shut up, or placed
in a Zorro II backplane.  It only responds if actually configured,
of course. This signal must only change at the end of a cycle, if
actually operating. */

CFGOUT              =  PRECON & CFGIN & !FCS & !DOE
                    #  PRECON & CFGOUT
                    #  Z2SHUNT & CFGIN;

/* This is the refresh acknowledge cycle.  When the a refresh request
comes in, and the coast is clear, this line is asserted to start
the refresh machine.  Determining when the coast is clear, eg,
arbitrating refresh, is the trick to all hand-made DRAM controllers.
This one works pretty simply.  The coast is clear when there's no
bus cycle happening, or when a bus cycle is happening but another
slave is responding.  The trick is avoid races; FCS could be
changing just as REFREQ comes in.  Therefore, the second half of
this arbiter is in the RAS cycle generation, which doesn't start
until REFACK is negated and SLAVE is asserted.  */

REFACK              = REFREQ & !FCS & !MATCH
                    # REFREQ &  FCS & !SLAVE & DOE
                    # REFACK &  REFREQ;

/* The multiple cycle transfer acknowledge.  If the jumper enables
them, and a refresh isn't already requested, we'll acknowledge
them.  If a refresh request comes in, we'll negate MTACK after
the current cycle finishes, which will result in one more
burst cycle before the full cycle terminates and the refresh
can be acknowledged.  I do it this way because I use the
refresh timer to handle the TRASMAX limitation of the DRAM as
wall as handling refresh. */

MTACK               = hit & BRENB & !REFREQ
                    # hit & MTACK & !DOE
                    # hit & MTACK &  MTCR;

MTACK.OE            = hit;

/* This is SLAVE, the board select line.  Most board activity centers
around this line.  If the board is selected and unconfigured,
always respond.  Once configured, only respond if it's not shutup
or shunted.  This line is held through the cycle's end. */

SLAVE               = select & FCS & CFGIN &  !CFGOUT
                    # select & FCS & CFGLT &   CFGOUT;

/* This indicates if the cycle is a burst cycle.  The first cycle is
always a non-burst cycle.  If, at the end of the first cycle,
MTCR and MTACK are asserted, all subsequent cycles are burst
until FCS is negated. */

BURST               = SLAVE & DOE & MTCR & MTACK
                    # BURST & FCS;
```

A.1.3 Memory Timing PAL

This device controls RAS and CAS timing, /DTACK generation, and high order RAM banking. This must be programmed into a 10ns 20L8 or equivalent device.

```
        PARTNO      U300 ;
        NAME        U300 ;
        DATE        January 10, 1991 ;
        REV         6 ;
        DESIGNER    Dave Haynie ;
        COMPANY     Commodore-Amiga ;
        ASSEMBLY    BIGRAM ;
        LOCATION    U300 ;

/************************************************************************/
/*                                                                      */
/* Zorro III BIGRAM DRAM Timing                                         */
/*                                                                      */
/*      This device controls the standard and refresh timing of the     */
/*      dynamic RAM. Big-Time asynchronicity ahead!  This also controls */
/*      banking within a CAS controlled memory bank.                    */
/*                                                                      */
/************************************************************************/
/*                                                                      */
/* DEVICE DATA:                                                         */
/*                                                                      */
/*      Device:                 20L8-10                                 */
/*      Clock:                  NONE                                    */
/*      Unused:                 NONE                                    */
/*                                                                      */
/************************************************************************/
/*                                                                      */
/* REVISION HISTORY:                                                    */
/*                                                                      */
/*      DBH 05-02: Original version.                                    */
/*      DBH 05-24: Hacked refresh logic to remove CAS bounce.           */
/*      DBH 05-24: More refresh logic adjustments.                      */
/*      DBH 05-24: Fixed refresh end-of-cycle.                          */
/*      DBH 05-30: Fixed slow CAS problems.                             */
/*                                                                      */
/*      DBH 01-10: Fixed burst RAS problem.                             */
/*      DBH 01-10: Fixed page-mode RAS/CAS generation.                  */
/*                                                                      */
/************************************************************************/

/* INPUTS: */

PIN 1   =           !RASDEL     ;       /* RAS strobe delay */
PIN 2   =           !MUX        ;       /* DRAM Address multiplexer */
PIN 3   =           !MTCR       ;       /* Multiple cycle request. */
PIN 4   =           !BURST      ;       /* We're in burst mode. */
PIN 5   =            DOE        ;       /* Data time */
PIN 6   =           !SLAVE      ;       /* The board is responding */
PIN 7   =           !REFACK     ;       /* We're servicing a refresh request */
PIN 8   =            SCRAM      ;       /* We're using static column RAM. */
PIN 9   =            A23        ;       /* System addresses */
PIN 10  =            A22        ;
PIN 11  =            A21        ;
PIN 13  =            A20        ;
PIN 14  =            MEG4       ;       /* 4 Meg parts? */
PIN 23  =           !CASDEL     ;       /* CAS strobe delay */

/* BIDIRECTIONALS: */

PIN 16  =           !CASEN      ;       /* CAS strobe enable */
PIN 17  =           !CASOUT     ;       /* CAS delay input */
PIN 18  =           !REFCAS     ;       /* CAS for refresh */
PIN 19  =           !REFCYC     ;       /* We're in a refresh cycle. */
PIN 20  =           !DTACK      ;       /* Data is valid on bus */
PIN 21  =           !RASEN      ;       /* RAS strobe enable */

/* OUTPUTS: */

PIN 22  =            BK0        ;       /* Small Bank bit 0 */
PIN 15  =            BK1        ;       /* Small Bank bit 1 */

/** OUTPUT TERMS: **/

/* The data valid signal.  Data is valid on the bus if we're not in a refresh cycle,
the board is selected, and something's happened.  The non-burst cycle is driven by RAS
delay only, the burst cycle by CAS delay only. */

DTACK               = SLAVE & !BURST & !REFACK & !REFCYC & DOE & RASDEL & CASDEL
                    # SLAVE & !BURST &  DTACK
                    # SLAVE &  BURST & !REFACK & !REFCYC & DOE & CASOUT & CASDEL & MTCR
                    # SLAVE &  BURST &  DTACK & MTCR;
```

```
DTACK.OE            = SLAVE;

/* The RAS enable strobe.  If we're not in refresh, it goes as soon
as we're sure the board is selected.  If refresh is called for,
start a RAS cycle after the CAS delay. */

RASEN               = !REFACK & !REFCYC & !RASDEL & !CASEN & SLAVE
                    # !REFACK & !REFCYC &  RASEN  &  SLAVE
                    #  REFCYC  &  CASDEL & !RASDEL
                    #  REFCYC  &   RASEN & !RASDEL;

/* The CAS enable works differently for burst vs. non-burst.  For non-burst,
it follows RASEN after DOE and MUX are asserted.  In a burst cycle, it
follows MTCR.  For refresh, CAS can't be enabled until we're sure that
RASDEL is negated, thus ensuring RAS precharge when a refresh cycle
immediately follows a standard memory cycle.  */

CASOUT              = !REFACK & !REFCYC & !BURST &  RASEN  &  MUX &   DOE & !RASDEL
                    # !REFACK & !REFCYC & !BURST &  CASOUT & SLAVE
                    # !REFACK & !REFCYC &  BURST & !CASDEL & MTCR
                    # !REFACK & !REFCYC &  BURST &  CASOUT & MTCR
                    #  REFACK &  REFCYC & !RASEN & !RASDEL
                    #  CASOUT &  REFCYC & !RASEN;

/* The actual CAS that goes out is modified by our use of SCRAMs.  If
SCRAMs are in use, CASEN goes low and stays low, while CASOUT works
the DTACK line.  Otherwise, CASEN and CASOUT are the same. */

CASEN               = !REFACK & !REFCYC & !SCRAM & CASOUT
                    # !REFACK & !REFCYC &  SCRAM & CASOUT
                    # !REFACK & !REFCYC &  SCRAM &  CASEN & SLAVE;

/* This is the rest of the refresh machine.  A refresh cycle starts with a
valid refresh acknowledge and the assertion of the standard and refresh
CAS.  RAS for refresh is asserted one CASDEL later, and standard CAS is
negated at the same point.  The refresh counter will clear REFREQ when
REFCYC is asserted, and clear REFACK when REFREQ is negated. */

REFCAS              = REFACK & REFCYC & !CASDEL & !RASDEL
                    # REFCAS & REFCYC & !MUX
                    # REFCAS & REFCYC &  RASEN & !RASDEL;

REFCYC              = REFACK & !CASDEL & !RASDEL
                    # REFCYC &  CASOUT & !RASDEL
                    # REFCYC &   RASEN
                    # REFCYC &  RASDEL;

/* Bank control. The bank is controlled by A23 and A22 for 4 Meg memory,
A21 and A20 for 1 Meg memory. */

BK0                 = A22 &  MEG4
                    # A20 & !MEG4;

BK1                 = A23 &  MEG4
                    # A21 & !MEG4;
```

A.1.4 CAS Control PAL

This device controls the CAS generation and banking. This must be programmed into a 15ns 20L8 PAL device or equivalent.

```
            PARTNO     U304 ;
            NAME       U304 ;
            DATE       May 30, 1990 ;
            REV        3 ;
            DESIGNER   Dave Haynie ;
            COMPANY    Commodore-Amiga ;
            ASSEMBLY   BIGRAM ;
            LOCATION   U304 ;

/************************************************************************/
/*                                                                      */
/*  Zorro III BIGRAM DRAM CAS Select                                    */
/*                                                                      */
/*      This device controls the CAS strobes, which control DRAM byte   */
/*      enables and most significant bank.                              */
/*                                                                      */
/************************************************************************/
/*                                                                      */
/*  DEVICE DATA:                                                        */
/*                                                                      */
/*      Device:                 20L8-15                                 */
/*      Clock:                  NONE                                    */
/*      Unused:                 NONE                                    */
/*                                                                      */
/************************************************************************/

/* INPUTS: */

PIN 1   =           !CASEN      ;       /* Normal CAS enable */
PIN 2   =           !DTACK      ;       /* Zorro III cycle termination */
PIN 3   =           !REFCAS     ;       /* CAS for refresh cycle */
PIN 4   =           !REFACK     ;       /* We're in refresh */
PIN 5   =           !DS3        ;       /* Zorro III data strobes */
PIN 6   =           !DS2        ;
PIN 7   =           !DS1        ;
PIN 8   =           !DS0        ;
PIN 9   =            SCRAM      ;       /* Using static column memories */
PIN 10  =            READ       ;       /* Zorro III Read enable */
PIN 11  =            MEG4       ;       /* Are we using 4 Meg parts? */
PIN 13  =           !CADDR      ;       /* Column Address Valid */
PIN 14  =            A24        ;       /* Address lines */
PIN 23  =            A22        ;

/* OUTPUTS: */

PIN 15  =           !CASL0      ;       /* Lower bank CAS */
PIN 16  =           !CASL1      ;
PIN 17  =           !CASL2      ;
PIN 18  =           !CASL3      ;
PIN 19  =           !CASH0      ;       /* Upper bank CAS */
PIN 20  =           !CASH1      ;
PIN 21  =           !CASH2      ;
PIN 22  =           !CASH3      ;

/** INTERNAL TERMS: **/

/* The CAS lines are the highest order banking control.  If we're using 1 Meg
parts, lower is $0000000-$03fffff, upper is $0400000-$07fffff, so A22 controls
the banking.  If we're using 4 Meg parts, lower is $0000000-$0ffffff, upper is
$1000000-$1ffffff, so A24 controls the banking.  */

lower               = !A24 &  MEG4 & CASEN & CADDR
                    # !A22 & !MEG4 & CASEN & CADDR;
upper               =  A24 &  MEG4 & CASEN & CADDR
                    #  A22 & !MEG4 & CASEN & CADDR;

/** OUTPUT TERMS: **/

/* The CAS terms are simple.  There are two banks of memory, and the banking
is controlled as above.  On writes, the data strobes control the particular
CAS line, and we wait for WRDEL so that data is guaranteed valid on the
DRAM bus.  On reads, all CAS lines in a bank are asserted ASAP.  On
refresh, all CAS lines are asserted. */

CASL0               = lower & !READ & DS0
                    # lower &  READ
                    # REFCAS;

CASL1               = lower & !READ & DS1
                    # lower &  READ
                    # REFCAS;
```

```
CASL2                   = lower & !READ & DS2
                        # lower &  READ
                        # REFCAS;

CASL3                   = lower & !READ & DS3
                        # lower &  READ
                        # REFCAS;

CASH0                   = upper & !READ & DS0
                        # upper &  READ
                        # REFCAS;

CASH1                   = upper & !READ & DS1
                        # upper &  READ
                        # REFCAS;

CASH2                   = upper & !READ & DS2
                        # upper &  READ
                        # REFCAS;

CASH3                   = upper & !READ & DS3
                        # upper &  READ
                        # REFCAS;
```

A.1.5 Refresh Counter PAL

This device is responsible for timing the CAS-before-RAS refresh used by the DRAM system. This must be programmed into a 25ns 16R8 or equivalent device.

```
       PARTNO     U306 ;
       NAME       U306 ;
       DATE       May 30, 1990 ;
       REV        1 ;
       DESIGNER   Dave Haynie ;
       COMPANY    Commodore-Amiga ;
       ASSEMBLY   BIGRAM ;
       LOCATION   U306 ;

/**********************************************************************/
/*                                                                    */
/* Zorro III BIGRAM DRAM Refresh Counter.                             */
/*                                                                    */
/*     This device is responsible for generating refresh request.     */
/*                                                                    */
/**********************************************************************/
/*                                                                    */
/* DEVICE DATA:                                                       */
/*                                                                    */
/*     Device:    16R8-25                                             */
/*     Clock:     C7M                                                 */
/*     Unused:    NONE                                                */
/*                                                                    */
/**********************************************************************/

/* INPUTS: */

PIN 2  =          !REFACK    ;      /* We're servicing a refresh request */
PIN 3  =          !REFCYC    ;      /* We're in a refresh cycle. */

/* BIDIRECTIONALS: */

PIN 19 =          !REFREQ    ;      /* Refresh request */

/* USED INTERNALLY: */

PIN 18 =          !R0        ;      /* Counter bits */
PIN 17 =          !R1        ;
PIN 16 =          !R2        ;
PIN 15 =          !R3        ;
PIN 14 =          !R4        ;
PIN 13 =          !R5        ;
PIN 12 =          !R6        ;

/** INTERNAL TERMS: **/

field  count      = [R6..0];

/** OUTPUT TERMS: **/

/* The refresh request is asserted when the terminal count has been reached.
It's held until REFHOLD is asserted. */


REFREQ.D          = count:70
                  # REFREQ & !REFCYC;

/* The refresh counter is pretty simple.  We're assuming one refresh cycle
every 15,625ns, which works out fine for the 8ms, 512 row 1 Meg parts
or the 16ms, 1024 row 4 Meg parts.  However, the maximum TRAS period
is only 10,000ns, which must be taken into account to support burst
mode.  Counting 71 140ns C7M clocks gets me to 9,940ns, close enough.
The counter resets when REFCYC comes along. */

R0.D              = !REFCYC & !R0;

R1.D              = !REFCYC &  R0 & !R1
                  # !REFCYC & !R0 &  R1;

R2.D              = !REFCYC &  R0 &  R1 & !R2
                  # !REFCYC &       !R1 &  R2
                  # !REFCYC & !R0       &  R2;

R3.D              = !REFCYC &  R0 &  R1 &  R2 & !R3
                  # !REFCYC &             !R2 &  R3
                  # !REFCYC &       !R1       &  R3
                  # !REFCYC & !R0             &  R3;

R4.D              = !REFCYC &  R0 &  R1 &  R2 &  R3 & !R4
                  # !REFCYC &                   !R3 &  R4
                  # !REFCYC &             !R2       &  R4
```

```
                    # !REFCYC &       !R1             &  R4
                    # !REFCYC & !R0                   &  R4;

R5.D                = !REFCYC &  R0 &  R1 &  R2 &  R3 &  R4 & !R5
                    # !REFCYC &                         !R4 &  R5
                    # !REFCYC &                   !R3       &  R5
                    # !REFCYC &             !R2             &  R5
                    # !REFCYC &       !R1                   &  R5
                    # !REFCYC & !R0                         &  R5;

R6.D                = !REFCYC &  R0 &  R1 &  R2 &  R3 &  R4 &  R5 & !R6
                    # !REFCYC &                               !R5 &  R6
                    # !REFCYC &                         !R4       &  R6
                    # !REFCYC &                   !R3             &  R6
                    # !REFCYC &             !R2                   &  R6
                    # !REFCYC &       !R1                         &  R6
                    # !REFCYC & !R0                               &  R6;
```

A.2 Schematics

The following pages contain the schematics for the example memory board. The list of parts is as follows:

Capacitors

0.01 µF MLC	C109
0.10 µF MLC	C100-C107,C200-C203,C300-C306,C400-C404
0.33 µF MLC	C500,C502,C504,C506,C508,C510,C512,C514,C516,C518,C520, C522,C524,C526,C528,C530,C600,C602,C604,C606,C608,C610, C612,C614,C616,C618,C620,C622,C624,C626,C628,C630
47 µF, 16V Electro	C108
100 µF, 16V Electro	C110,C111

Resistors

22 Ω, 5%, 1/4 Watt	R300,R301
1K Ω, 5%, 1/4 Watt	R100

Resistor Packs

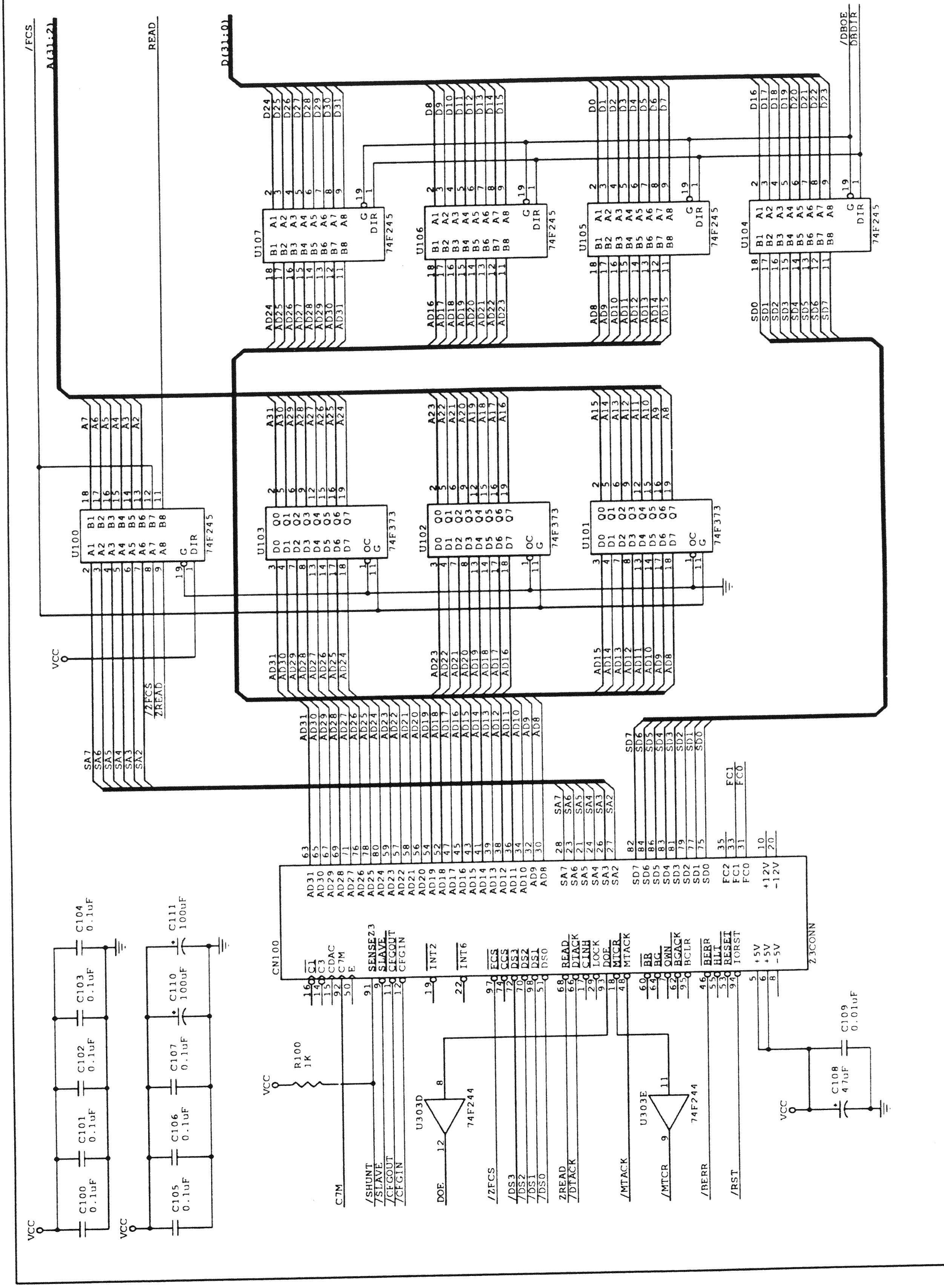
22 Ω, 4x8	RP300-RP303,RP400-RP404
1K Ω, 9x10	RP100

Post Jumpers

3 Pin, 0.100	J200,J302,J303
2 Pin x 3 Pin, 0.100	J300
2 Pin x 4 Pin, 0.100	J301

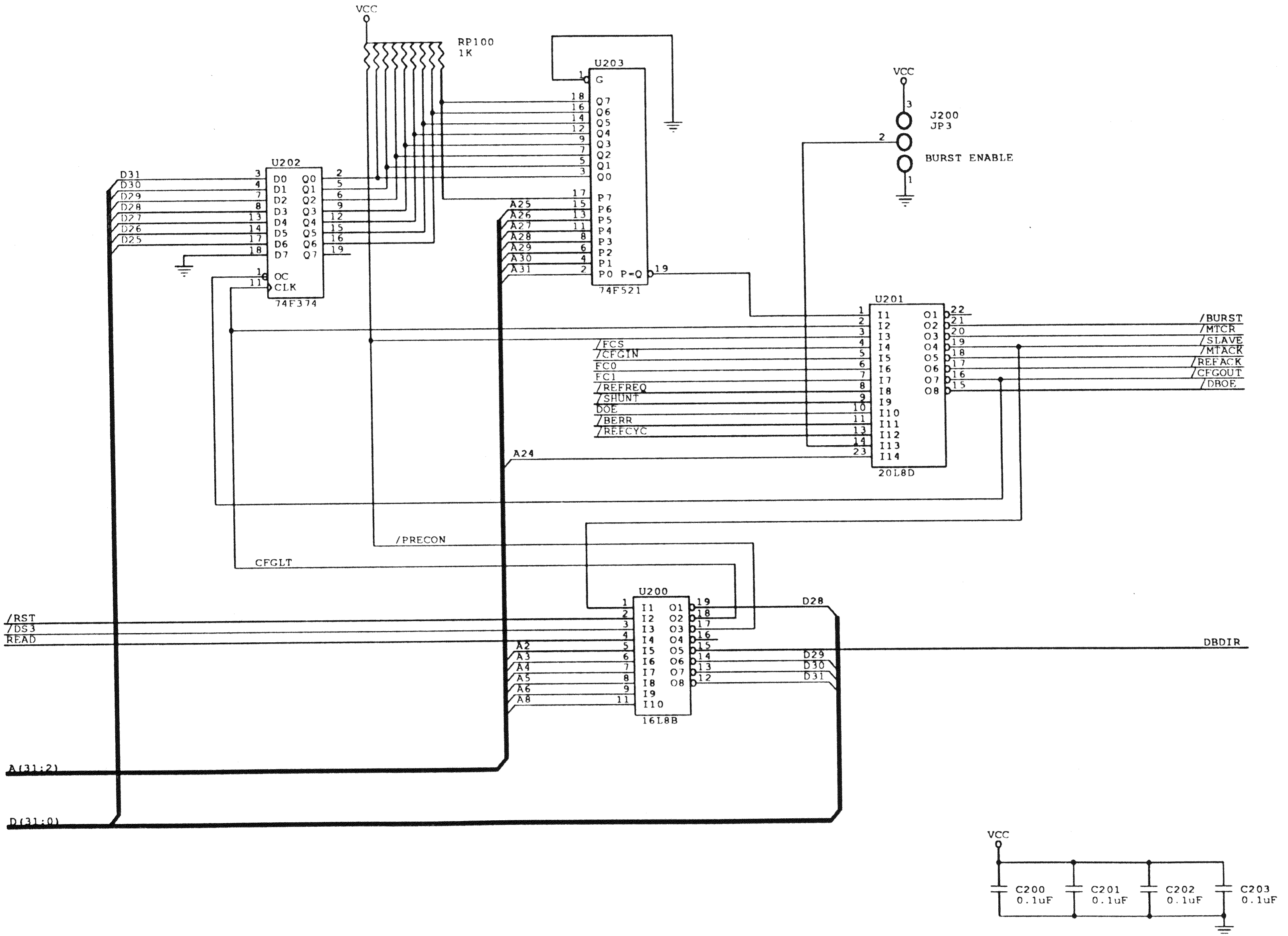
Integrated Circuits

74F138	U305
74F244	U303
74F245	U100,U104-U107
74F258	U400-U404
74F373	U101-U103
74F374	U202
74F521	U203
PAL 16L8B	U200
PAL 16R8A	U306
PAL 20L8B	U300,U304
PAL 20L8D	U201
Tap Delay 100ns	U301
Tap Delay 50ns	U302
DRAM 256K x 4, 80ns or 1M x 4, 80ns	U500-U531,U600-U631

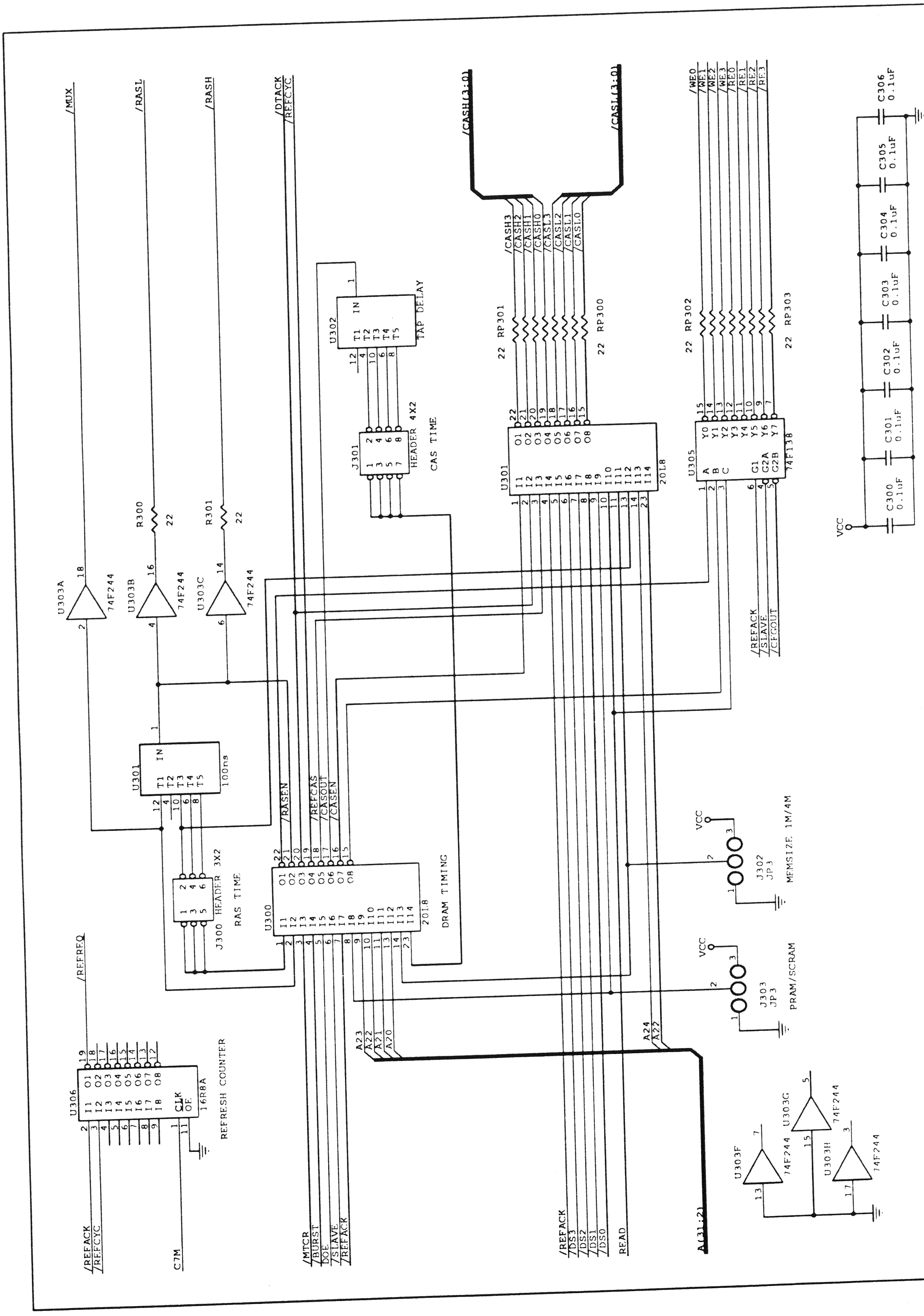


/FCS
A(31:2)
READ
D(31:0)
/DBOE
DBDIR
U100
U101
U102
U103
U104
U105
U106
U107
74F245
74F373
CN100
Z3CONN
U303D
U303E
74F244
R100 1K
VCC
C100 0.1uF
C101 0.1uF
C102 0.1uF
C103 0.1uF
C104 0.1uF
C105 0.1uF
C106 0.1uF
C107 0.1uF
C108 47uF
C109 0.01uF
C110 100uF
C111 100uF
C7M
/SHUNT
/SLAVE
/CFGOUT
/CFGIN
DOE
/ZFCS
/DS3
/DS2
/DS1
/DS0
ZREAD
/DTACK
/MTACK
/MTCR
/BERR
/RST
/ZFCS
ZREAD

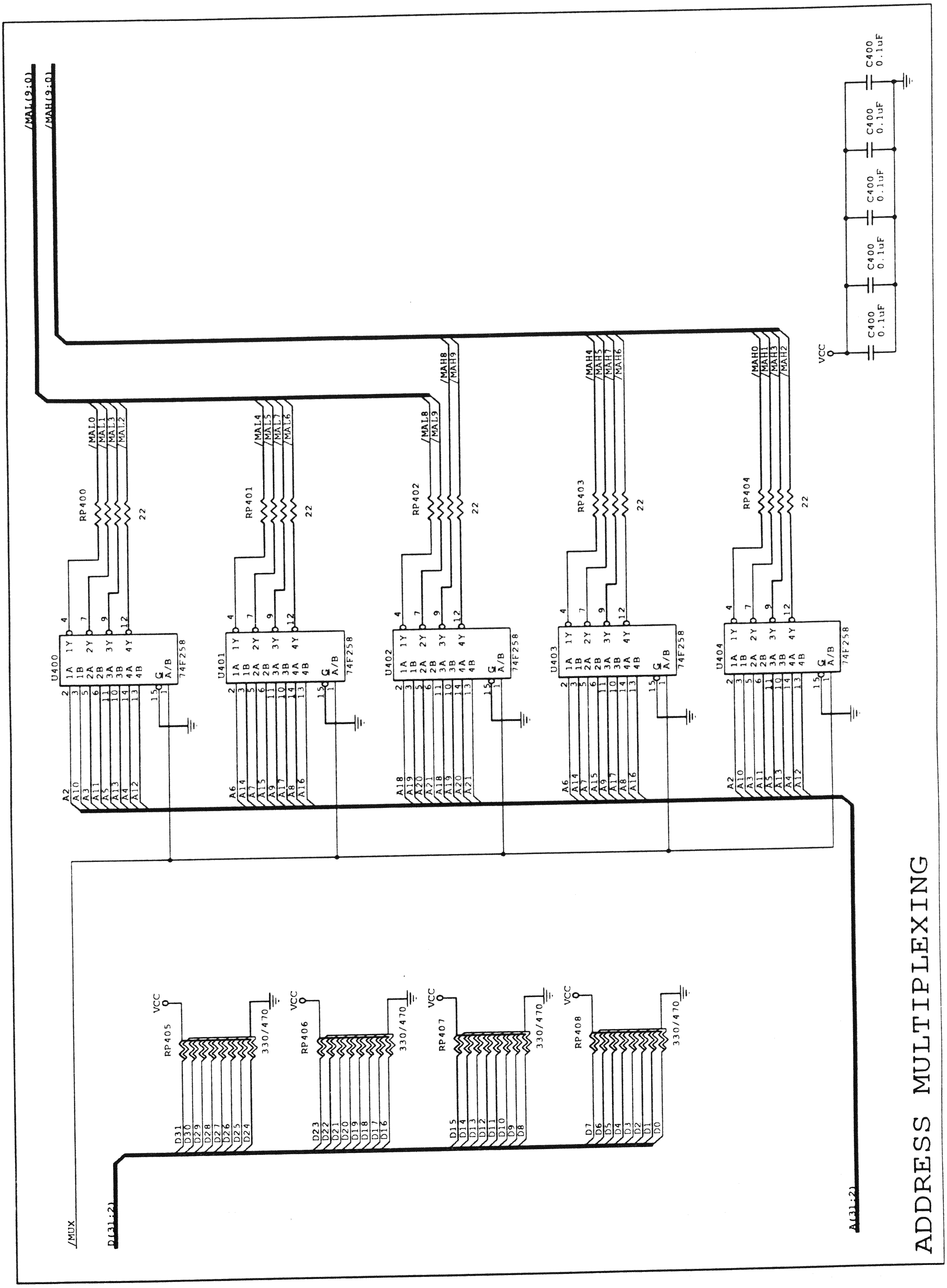
BUS BUFFERS & LATCHES

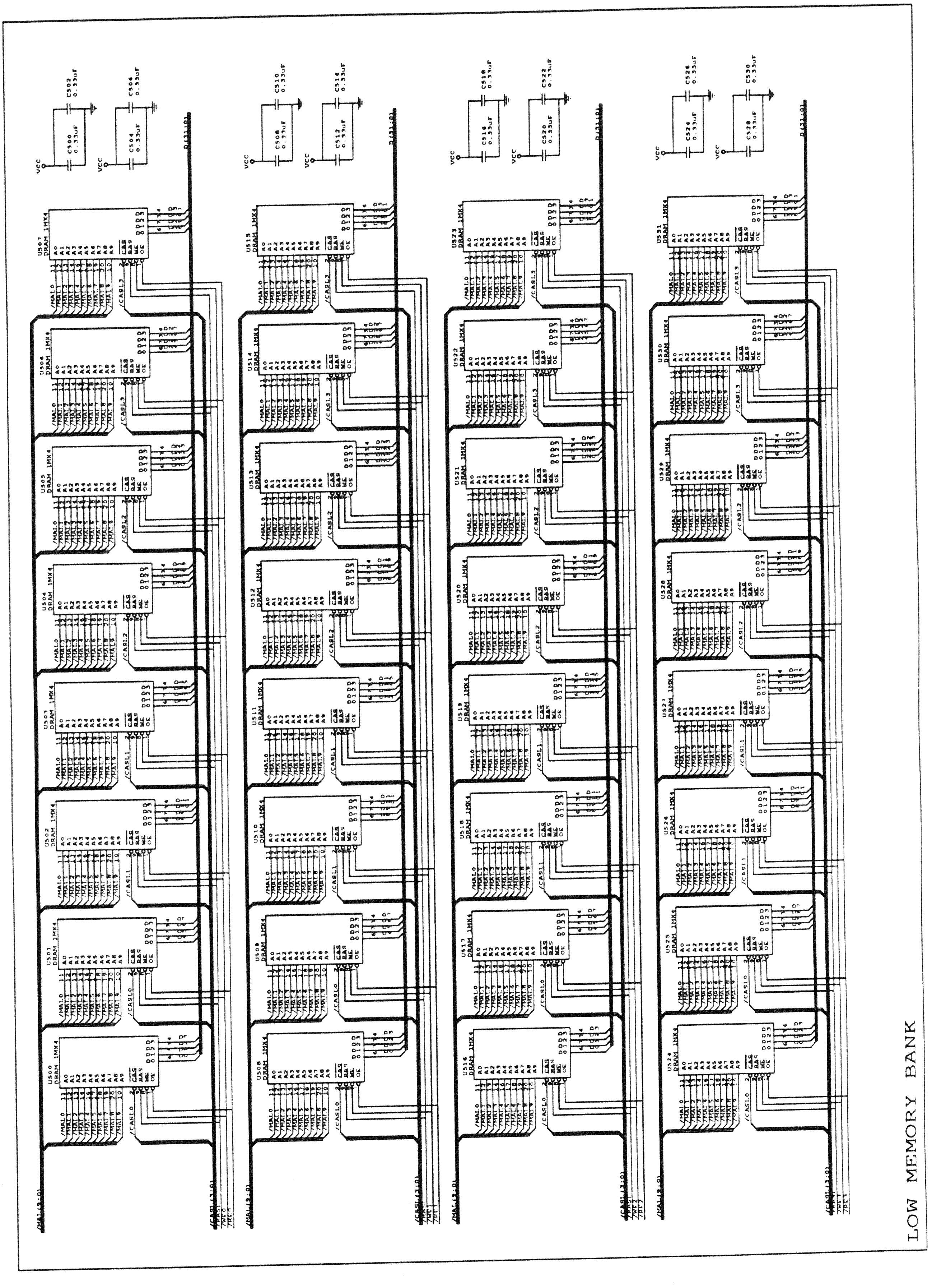

VCC
RP100
1K
U203
G
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0
P7
P6
P5
P4
P3
P2
P1
P0
P=Q
74F521
U202
D0
D1
D2
D3
D4
D5
D6
D7
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
OC
CLK
74F374
D31
D30
D29
D28
D27
D26
D25
A25
A26
A27
A28
A29
A30
A31
A24
J200
JP3
BURST ENABLE
U201
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
I11
I12
I13
I14
O1
O2
O3
O4
O5
O6
O7
O8
20L8D
/FCS
/CFGIN
FC0
FC1
/REFREQ
/SHUNT
DOE
/BERR
/REFCYC
/BURST
/MTCR
/SLAVE
/MTACK
/REFACK
/CFGOUT
/DBOE
/PRECON
CFGLT
U200
16L8B
/RST
/DS3
READ
A2
A3
A4
A5
A6
A8
D28
D29
D30
D31
DBDIR
A(31:2)
D(31:0)
C200
0.1uF
C201
0.1uF
C202
0.1uF
C203
0.1uF

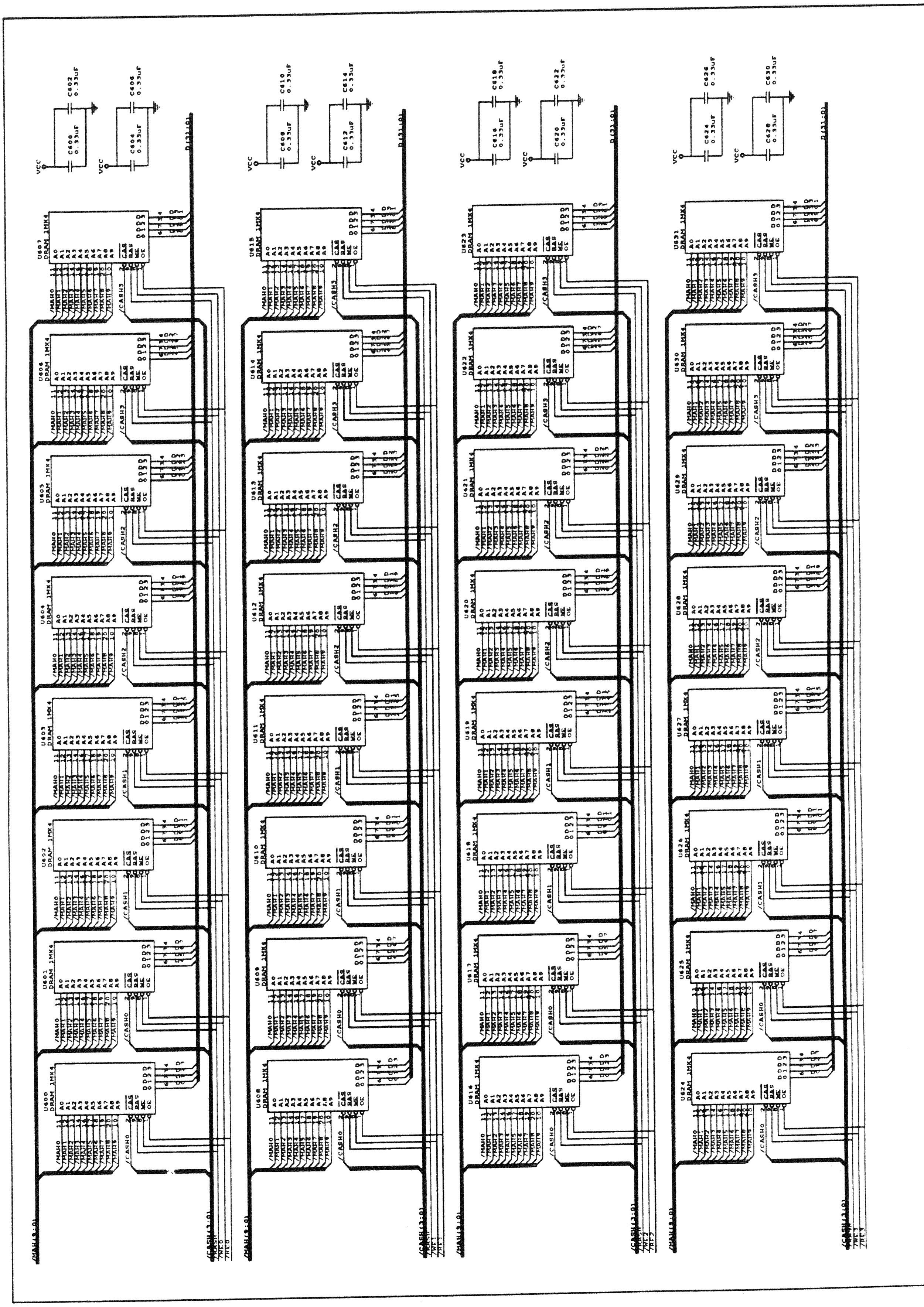
CONFIGURATION & BUFFER CONTROL


/MUX
/RASL
/RASH
/DTACK
/REFCYC
/CASH(3:0)
/CASL(3:0)
/CASH3
/CASH2
/CASH1
/CASH0
/CASL3
/CASL2
/CASL1
/CASL0
/WE0
/WE1
/WE2
/WE3
/RE0
/RE1
/RE2
/RE3
C306 0.1uF
C305 0.1uF
C304 0.1uF
C303 0.1uF
C302 0.1uF
C301 0.1uF
C300 0.1uF
VCC
U302
TAP DELAY
22 RP301
22 RP300
22 RP302
22 RP303
J301
HEADER 4X2
CAS TIME
U301
20L8
U305
74F138
U303A
U303B
U303C
74F244
R300
R301
22
/REFACK
/SLAVE
/CFGOUT
U301
100ns
/RASEN
/REFCAS
/CASOUT
/CASEN
J300
HEADER 3X2
RAS TIME
U300
20L8
DRAM TIMING
J302
JP2
MEMSIZE 1M/4M
J303
JP3
PRAM/SCRAM
/REFREQ
U306
16R8A
REFRESH COUNTER
CLK
OE
A23
A22
A21
A20
A24
A22
U303F
U303G
U303H
/REFACK
/REFCYC
C7M
/MTCR
/BURST
/OE
/SLAVE
/REFACK
/REFACK
/DS3
/DS2
/DS1
/DS0
READ
A(31:2)

MEMORY CONTROL & REFRESH MACHINE


ADDRESS MULTIPLEXING
/MAL(9:0)
/MAH(9:0)
/MAL0
/MAL1
/MAL3
/MAL2
/MAL4
/MAL5
/MAL7
/MAL6
/MAL8
/MAL9
/MAH8
/MAH9
/MAH4
/MAH5
/MAH7
/MAH6
/MAH0
/MAH1
/MAH3
/MAH2
RP400
RP401
RP402
RP403
RP404
22
U400
U401
U402
U403
U404
74F258
1A
1B
2A
2B
3A
3B
4A
4B
1Y
2Y
3Y
4Y
G
A/B
A2
A10
A3
A11
A5
A13
A4
A12
A6
A14
A7
A15
A9
A17
A8
A16
A18
A19
A20
A21
/MUX
D(31:2)
A(31:2)
VCC
RP405
RP406
RP407
RP408
330/470
D31
D30
D29
D28
D27
D26
D25
D24
D23
D22
D21
D20
D19
D18
D17
D16
D15
D14
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
C400
0.1uF


LOW MEMORY BANK
U500 DRAM 1MX4
U501 DRAM 1MX4
U502 DRAM 1MX4
U503 DRAM 1MX4
U504 DRAM 1MX4
U505 DRAM 1MX4
U506 DRAM 1MX4
U507 DRAM 1MX4
U508 DRAM 1MX4
U509 DRAM 1MX4
U510 DRAM 1MX4
U511 DRAM 1MX4
U512 DRAM 1MX4
U513 DRAM 1MX4
U514 DRAM 1MX4
U515 DRAM 1MX4
U516 DRAM 1MX4
U517 DRAM 1MX4
U518 DRAM 1MX4
U519 DRAM 1MX4
U520 DRAM 1MX4
U521 DRAM 1MX4
U522 DRAM 1MX4
U523 DRAM 1MX4
U524 DRAM 1MX4
U525 DRAM 1MX4
U526 DRAM 1MX4
U527 DRAM 1MX4
U528 DRAM 1MX4
U529 DRAM 1MX4
U530 DRAM 1MX4
U531 DRAM 1MX4
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9
CAS RAS WE OE
D0 D1 D2 D3
/MAL0 /MAL1 /MAL2 /MAL3 /MAL4 /MAL5 /MAL6 /MAL7 /MAL8 /MAL9
/CASL0
/CASL1
/CASL2
/CASL3
/MAL(9:0)
/CASL(3:0)
/WE0
/WE1
D(31:0)
VCC
C500 0.33uF
C502 0.33uF
C504 0.33uF
C506 0.33uF
C508 0.33uF
C510 0.33uF
C512 0.33uF
C514 0.33uF
C516 0.33uF
C518 0.33uF
C520 0.33uF
C522 0.33uF
C524 0.33uF
C526 0.33uF
C528 0.33uF
C530 0.33uF


DRAM 1MX4
U600
U601
U602
U603
U604
U605
U606
U607
U608
U609
U610
U611
U612
U613
U614
U615
U616
U617
U618
U619
U620
U621
U622
U623
U624
U625
U626
U627
U628
U629
U630
U631
VCC
0.33uF
D[31:0]
/MAH[9:0]
/CASH[3:0]
/CASH0
/CASH1
/CASH2
/CASH3

HIGH MEMORY BANK

A.3 Zorro III Configuration

While AmigaOS 2.0 understands Zorro III AUTOCONFIG® conventions, the following routine is useful for configuring simple Zorro III boards in an AmigaOS 1.3 system. Note that many popular MMU configurations don't map in the Zorro III configuration space at $FF000000, so this program is not likely to work with an MMU mapping in place.

```
/* ====================================================================== */

/* A very simple configuration utility for Zorro III boards.  This code will
   configure Zorro III cards that are placed after any Zorro II cards in
   the A3000.  All configuration is done based on 16 meg slots and no magic
   for autoboot, etc.  */

#include <exec/types.h>
#include <exec/memory.h>
#include <libraries/configregs.h>
#include <libraries/configvars.h>
#include <libraries/expansionbase.h>
#include <stdio.h>
#include <ctype.h>
#include <functions.h>

/* ====================================================================== */

/* Modified configuration information. */

/* Extensions to the TYPE field. */

#define E_Z3EXPBASE        0xff000000L
#define E_Z3EXPSTART       0x10000000L
#define E_Z3EXPFINISH      0x7fffffffL
#define E_Z3SLOTSIZE       0x01000000L
#define E_Z3ASIZEINC       0x00010000L

#define ERT_ZORROII        ERT_NEWBOARD
#define ERT_ZORROIII       0x80

/* Extensions to the FLAGS field. */

#define ERFB_EXTENDED      5L
#define ERFF_EXTENDED      (1L<<5)

static BoardSize[2][8] = {
    { 0x00800000,0x00010000,0x00020000,0x00040000,
      0x00080000,0x00100000,0x00200000,0x00400000 },
    { 0x01000000,0x02000000,0x04000000,0x08000000,
      0x10000000,0x20000000,0x40000000,0x00000000 }
};

#define ERFB_QUICKVALID    4L
#define ERFF_QUICKVALID    (1L<<4)

#define ERF_SUBMASK        0x0fL
#define ERF_SUBSAME        0x00L
#define ERF_SUBAUTO        0x01L
#define ERF_SUBFIXED       0x02L
#define ERF_SUBRESERVE     0x0eL

static SubSize[16] = {
    0x00000000,0x00000000,0x00010000,0x00020000,
    0x00040000,0x00080000,0x00100000,0x00200000,
    0x00400000,0x00600000,0x00800000,0x00a00000,
    0x00c00000,0x00e00000,0x00000000,0x00000000
};

#define PRVB(x)if (verbose) { printf(x); }

static BOOL                verbose = TRUE;
static BOOL                anyone = FALSE;
struct ExpansionBase       *ExpansionBase;
static ULONG               Z3Space = 0x10000000L;

/* ====================================================================== */

/* These functions are involved in finding a Zorro III board. */

/* This function reads the logical value stored at the given Zorro III
   ROM location.  This corrects for complements and the differing offsets
   depending on location. */
```

```
UBYTE ReadZ3Reg(base,reg)
WORD *base;
WORD reg;
{
    ULONG *Z3base;
    UWORD result;

    if (base == (WORD *)E_EXPANSIONBASE) {
        base += (reg>>1);
        result = ((*base++)&0xf000)>>8;
        result = ((*base)&0xf000)>>12;
    } else {
        Z3base = (ULONG *)(base+(reg>>1));
        result = ((*Z3base)&0xf0000000)>>24;
        result |= ((*(Z3base+0x40))&0xf0000000)>>28;
    }
    if (reg) result = ~result;

    return (UBYTE)result;
}

/* This function types the board in the system, returning the type code.
    There are four possibilities -- no board, a Zorro II board, a Zorro III
    board at the Zorro II configuration slot, and a Zorro III board at the
    Zorro III configuration slot. */

#define BT_NONE         0
#define BT_Z2           1
#define BT_Z3_AT_Z2     2
#define BT_Z3_AT_Z3     3

BYTE TypeOfPIC() {
    UBYTE type;
    UWORD manf;

    type = ReadZ3Reg(E_EXPANSIONBASE,0x00);
    manf = ReadZ3Reg(E_EXPANSIONBASE,0x10)<<8 | ReadZ3Reg(E_EXPANSIONBASE,0x14);

    if (manf != 0x0000 && manf != 0xffff) {
        if ((type & ERT_TYPEMASK) == ERT_ZORROII) return BT_Z2;
        if ((type & ERT_TYPEMASK) == ERT_ZORROIII) return BT_Z3_AT_Z2;
    }
    type = ReadZ3Reg(E_Z3EXPBASE,0x00);
    manf = ReadZ3Reg(E_Z3EXPBASE,0x10)<<8 | ReadZ3Reg(E_Z3EXPBASE,0x14);

    if (manf != 0x0000 && manf != 0xffff)
        if ((type & ERT_TYPEMASK) == ERT_ZORROIII) return BT_Z3_AT_Z3;

    return BT_NONE;
}

/* This function fills the configuration ROM field of the given
    ConfigDev, form the given address, based on the appropriate mapping
    rules. */

void InitZ3ROM(base,cd)
WORD *base;
struct ConfigDev *cd;
{
    struct ExpansionRom *rom;

    rom = &cd->cd_Rom;

    rom->er_Type = ReadZ3Reg(base,0x00);
    rom->er_Product = ReadZ3Reg(base,0x04);
    rom->er_Flags = ReadZ3Reg(base,0x08);
    rom->er_Reserved03 = ReadZ3Reg(base,0x0c);
    rom->er_Manufacturer = ReadZ3Reg(base,0x10)<< 8 | ReadZ3Reg(base,0x14);
    rom->er_SerialNumber = ReadZ3Reg(base,0x18)<<24 | ReadZ3Reg(base,0x1c)<<16 |
    ReadZ3Reg(base,0x20)<< 8 | ReadZ3Reg(base,0x24);
    rom->er_InitDiagVec = ReadZ3Reg(base,0x28)<< 8 | ReadZ3Reg(base,0x2c);
    rom->er_Reserved0c = ReadZ3Reg(base,0x30);
    rom->er_Reserved0d = ReadZ3Reg(base,0x34);
    rom->er_Reserved0e = ReadZ3Reg(base,0x38);
    rom->er_Reserved0f = ReadZ3Reg(base,0x3c);
}

/* This function locates a Zorro III board.  If it finds one in the
    unconfigured state, it allocates a ConfigDev for it, fills in the
    configuration data, and returns that ConfigDev.  Otherwise it returns
    NULL.  It knows the basics of what to do should it encounter a
    Zorro II board sitting in the way. */

struct ConfigDev *FindZ3Board() {
    struct ConfigDev *cd;

    while (TRUE) {
        if (!(cd = AllocConfigDev())) return NULL;
        switch (TypeOfPIC()) {
            case BT_NONE        :
                FreeConfigDev(cd);
                return NULL;
            case BT_Z2          :
                PRVB("FOUND: Z2 Board, Configuring");
                if (!ReadExpansionRom(E_EXPANSIONBASE,cd))
                    if (!ConfigBoard(E_EXPANSIONBASE,cd))
                        AddConfigDev(cd);
                anyone = TRUE;
                break;
            case BT_Z3_AT_Z2    :
                PRVB("FOUND: Z3 Board (Z2 Space), Configuring");
                InitZ3ROM(E_EXPANSIONBASE,cd);
                cd->cd_BoardAddr = (APTR)E_EXPANSIONBASE;
                anyone = TRUE;
                return cd;
            case BT_Z3_AT_Z3    :
                PRVB("FOUND: Z3 Board (Z3 Space), Configuring");
                InitZ3ROM(E_Z3EXPBASE,cd);
                cd->cd_BoardAddr = (APTR)E_Z3EXPBASE;
                anyone = TRUE;
                return cd;
        }
    }
    return NULL;
}

/* ====================================================================== */

/* These functions are involved in configuring a Zorro III board. */

/* This function writes the configuration address stored in the given
   ConfigDev to the board in the proper way. */

void WriteCfgAddr(base,cd)
UWORD *base;
struct ConfigDev *cd;
{
    UBYTE nybreg[4],bytereg[2],*bytebase;
    UWORD wordreg,i,*wordbase;

    wordreg = (((ULONG)cd->cd_BoardAddr)>>16);
    bytereg[0] = (UBYTE)(wordreg & 0x00ff);
    bytereg[1] = (UBYTE)(wordreg >> 8);
    nybreg[0] = ((bytereg[0] & 0x0f)<<4);
    nybreg[1] = ((bytereg[0] & 0xf0));
    nybreg[2] = ((bytereg[1] & 0x0f)<<4);
    nybreg[3] = ((bytereg[1] & 0xf0));

    bytebase = (UBYTE *)(base + 22);
    wordbase = (UWORD *)(base + 22);

    if (base == (UWORD *)E_EXPANSIONBASE) {
        (*(bytebase+0x002)) = nybreg[2];
        (*(bytebase+0x000)) = bytereg[1];
        (*(bytebase+0x006)) = nybreg[1];
        (*(bytebase+0x004)) = bytereg[0];
    } else {
        (*(bytebase+0x104)) = nybreg[0];
        (*(bytebase+0x004)) = bytereg[0];
        (*(bytebase+0x100)) = nybreg[2];
        (*(wordbase+0x000)) = wordreg;
    }
}


/* This function automatically sizes the configured board described by the
   given ConfigDev.  It doesn't attempt to preserve the contents. */

void AutoSizeBoard(cd)
struct ConfigDev *cd;
{
    ULONG i,realmax,logicalsize = 0;

    realmax = ((ULONG)cd->cd_SlotSize) * E_Z3SLOTSIZE + (ULONG)cd->cd_BoardAddr;

    for (i = (ULONG)cd->cd_BoardAddr; i < realmax; i += E_Z3ASIZEINC)
        *((ULONG *)i) = 0;

    for (i = (ULONG)cd->cd_BoardAddr; i < realmax; i += E_Z3ASIZEINC) {
        if (*((ULONG *)i) != 0) break;
        *((ULONG *)i) = 0xaa5500ff;
        if (*((ULONG *)i) != 0xaa5500ff) break;
        logicalsize += E_Z3ASIZEINC;
    }
    cd->cd_BoardSize = (APTR)logicalsize;
}
```

```
/* This function configures a Zorro III board, based on the initialization
    data in its ConfigDev structure. */

void ConfigZ3Board(cd)
struct ConfigDev *cd;
{
    APTR base = cd->cd_BoardAddr;
    UWORD sizecode,extended,subsize;
    ULONG physsize,logsize;
    char *memname;

    /* First examine the physical sizing of the board. */

    sizecode = cd->cd_Rom.er_Type & ERT_MEMSIZE;
    extended = ((cd->cd_Rom.er_Flags & ERFF_EXTENDED) != 0);

    physsize = BoardSize[extended][sizecode];

    cd->cd_BoardAddr = (APTR)Z3Space;
    cd->cd_BoardSize = (APTR)physsize;
    cd->cd_SlotAddr = (Z3Space-E_Z3EXPSTART)/E_Z3SLOTSIZE;
    cd->cd_SlotSize = ((physsize/E_Z3SLOTSIZE)>0)?(physsize/E_Z3SLOTSIZE):1;
    Z3Space += cd->cd_SlotSize * E_Z3SLOTSIZE;

    /* Next, process the sub-size, if any. */

    if (subsize = (cd->cd_Flags & ERF_SUBMASK))
        cd->cd_BoardSize = (APTR)SubSize[subsize];

    if (verbose) {
        printf("  BOARD STATS:");
        printf("                  ADDRESS: $%lx",cd->cd_BoardAddr);
        if (cd->cd_BoardSize)
            printf("                     SIZE: $%lx",cd->cd_BoardSize);
        else
            printf("                     SIZE: AUTOMATIC => ");
    }

    /* Now, configure the board. */

    WriteCfgAddr(base,cd);
    if (!cd->cd_BoardSize) {
        AutoSizeBoard(cd);
        printf("$%lx",cd->cd_BoardSize);
    }

    if (cd->cd_BoardSize && (cd->cd_Rom.er_Type & ERTF_MEMLIST)) {
        strcpy(memname = (char *)AllocMem(20L,MEMF_CLEAR),"Zorro III Memory");
        AddMemList(cd->cd_BoardSize,MEMF_FAST|MEMF_PUBLIC,10,cd->cd_BoardAddr,memname);
    }

    AddConfigDev(cd);
}

/* ====================================================================== */

/* This is the main program. */

void main(argc,argv)
int argc;
char *argv[];
{
    int i;
    struct ConfigDev *cd;

    if (!(ExpansionBase = (struct ExpansionBase *)OpenLibrary("expansion.library",0L))) {
        printf("Error: Can't open "expansion.library"");
        exit(10);
    }
    if (argc > 1)
        for (i = 1; i < argc; ++i) switch (toupper(argv[i][0])) {
            case 'Q':  verbose = FALSE;   break;
            case 'V':  verbose = TRUE;    break;
    }

    while (cd = FindZ3Board()) ConfigZ3Board(cd);

    if (!anyone) PRVB("No PICs left to configure");
    CloseLibrary((struct ExpansionBase *)ExpansionBase);
}
```

Developing Network Applications for the Amiga

Dale Larson--Software Engineer, Amiga Networking Group
Siberia Office

1 Introduction

When you run a wire between two or more computers, you have a network. Big Deal. When your applications use that wire, however, you have a revolution.

1.1 Network Applications

Though some of the software is only internal or experimental, these are things I can do now with my Amiga, with software that I have now:

- From my Amiga, I can transparently access filesystems on Suns, on cbmvax, and on other Amigas.
- Whenever I print, I send my files to a network printer.
- I continuously receive mail on my Amiga--from as far away as Seattle, Sydney and Denmark, and from as near as a desk next to mine.
- Every night when I go home, my Amigas at work (and several others) are used by Chris Green to do distributed graphics rendering. The process is started over the network and all data is sent over the network. A picture that would have taken a week can be finished overnight.
- When I wish to use cbmvax or one of our Suns, I *rlogin* into it. I never access a physical terminal for other machines on our network.
- With a distributed windowing system, applications running on another machine can open windows on, and render graphics to, my machine. The standard distributed windowing system, X, is available for the Amiga from GfxBase. Many applications take advantage of X directly and are not otherwise network-aware.

In the scheme of what is possible, this is only the tip of the iceberg.

- A high school classroom with several computers could allow students to interactively participate in a simulation. It could allow them to collaborate electronically. It could allow teachers to electronically monitor and assist students. It would allow more and better computers into the classrooms by reducing the cost--peripherals such as printers, hard-drives and CD-ROMs could be more easily shared. Even the computational power of one more expensive machines could be shared by the students.

- Even a small office might use an email-like facility for phone messages and other notes. Another application might replace the intercom in a less intrusive way. Form letters might be kept in a central database accessed by a word processor. A distributed appointment calendar could allow a secretary to add a new appointment even as the boss is looking at his afternoon schedule. A distributed database application would allow access to such things as a central client database, inventory or order database. Maintaining the small office systems could be made dramatically easier with a network by allowing changes to propagate to other machines on the network.

- Multi-player interactive games have already been developed for the BBS systems, commercial information services and multi-user computers, most of which are run on over-taxed computers and are accessed at low speeds with character terminals. Imagine the games that could be created to use the computational power of each machine used to play the game and with the high speeds of a LAN.

- A software development environment might have several programmers working at the same time, updating the same sources. A network application eliminates the possibility of conflicts and the requirement for endless floppy-passing (RCS and a network filesystem can be used this way now). When a program is compiled, it could be started, via the network, on one or several test machines. Debugging information could be sent over the network, or a debugger on one machine could control the programs on others. (*Wack*, for example, has been hacked to run on a network).

- Multi-media applications might do any number of exciting things with the network. A few of the applications which have been experimented with on other machines are: real-time audio and video conferencing, interactive demos for groups, and shared electronic blackboards. The possibilities in this emerging field are boundless.

In much the same way as all applications are candidates for a GUI interface, all applications are candidates for becoming network applications. The GUI has only changed the ways in which people interact with their computers. Networks will change the ways in which people interact with each other.

1.2 Amiga Standards

Network applications should always be written to the Application Programmer's Interface (API) of some protocol, not to the network hardware. Network standards usually include several protocols layered one on top of another. These groups are often referred to as protocol stacks. At the lowest level, one of the protocols must interface to some network hardware. Fortunately for Amiga developers, Commodore has standardized a software interface to Amiga network hardware. This standard is the SANA-II Network Device Driver Specification (see Jesup (1991)). Many third-party hardware developers and protocol stack developers are committed to this standard. We are already testing SANA-II device drivers for various network hardware of our own, and should be testing at least one protocol stack written on top of SANA-II by the time you read this.

Unfortunately, we have not yet standardized an API to network protocol stacks (The SANA Protocol Device proposal discussed at last year's DevCon has not been finalized and is not being actively developed or endorsed by Commodore. The SANA Network Multiplexor Device proposal discussed at last year's DevCon is, of course, superceded by SANA-II). What this really means to you as an application developer is that you must choose the protocol to which your application will be written (with whatever interface is provided for that protocol). If that protocol uses SANA-II devices to interface to the hardware, though, your application will work on a wide variety of hardware. If you are careful to keep your protocol-specific code isolated and encapsulated (like you should be doing anyway), it should also be easy for you to make your code work with other network protocols.

The only protocol stack for the Amiga which is currently available from Commodore is TCP/IP. Our AS225 software package includes the standard TCP/IP protocols and several standard Internet applications. It has the same API as most Unix machines running TCP/IP, the Berkeley Sockets interface. Thousands of network applications have been written to the socket interface. AS225 has been shipping since December, 1990, and everything needed to write network applications for AS225 is included on the DevCon disks.

1.3 About This Paper

This paper will introduce you to the principles of writing network programs generally and to AS225's Berkeley Socket interface particularly. Even if you are writing to a different protocol stack with a different API, you should find the general principles in this paper helpful and the socket-specific particulars to be useful in making you aware of some portability issues you may face. It is expected that you have some familiarity with some protocol stack, preferably TCP/IP (see Hunt (1990) and/or Comer (1991a)). It will also be very helpful to be familiar with some networking package as a whole, preferably BSD or AS225.

If you are already familiar with writing network programs on Unix machines, you should feel right at home with the Shared Socket Library (In this paper, the Amiga *socket.library* is referred to as the Shared Socket Library to avoid confusion with *socket.lib*, the obsolete Amiga socket link library).

This paper is only an introduction, and the includes and Autodocs do not stand on their own. Even more so than in most of software development, networking seems simple in theory, but, in reality, gets complicated in a hurry. You should plan to pursue the material in the **References** section if you wish to develop network applications.

2 Minimalist Network Primer

If you are lucky, you may be able to get away with using this section as your only introduction to networking and network protocols. Hopefully it serves only as a refresher or a first introduction.

2.1 Protocol Layers

Recall that network protocols come in stacks. There are several layers, with hardware at the bottom and each layer above providing more functionality than those below. These layers are named in different ways by different people. The International Standards Organization (ISO) has created a reference-model to which all other layerings can be compared and which is used in all modem texts. The ISO 7-Layer Reference Model of Open Systems Interconnection looks like this:

Layer	Functionality
7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data Link (Hardware Interface)
1	Physical Hardware Connection

On the Amiga, layers 1 and 2 should always be the network hardware and SANA-II Network Device Driver. In AS225, layer 3 is the IP and ICMP protocols of the TCP/IP protocol stack. These protocols aren't used directly by application developers, suffice it to say that they handle machine to machine communication. The transport layer uses the network layer (and the layers below it) to provide communication between individual processes on different machines. Most current network applications use transport protocols. The transport protocols in TCP/IP are TCP and UDP (In the ISO world-view, the link layer and all protocols above it should be reliable. Since UDP is not a reliable protocol, it is not a transport protocol in a strict ISO interpretation). TCP provides reliable, stream-oriented communications, while UDP provides datagram communication (a datagram is a fixed-length message which isn't reliably delivered).

Layers 5 and 6, the presentation and session layers, do not exist in TCP/IP, or most other protocol stacks. So with TCP/IP on the Amiga, the protocol stack should look like this:

Application
Transport (TCP, UDP)
Network (IP, ICMP)
Data Link (SANA-II Network Device Driver)
Hardware (ethernet, arcnet, etc.)

In spite of the fact that protocols come in a stack, your application will only come into direct contact with a protocol at the top of the stack. In theory, you are not required to know protocols below the one used for your application. In practice, higher-level protocols are often described in terms of additions to lower-level protocols.

2.2 Addresses

All data on the network is sent to and from network addresses. There are many different types of network addresses, at least one type for each layer of the protocol stack. Applications generally send data over the network with a transport-layer protocol, hence the data is sent from one transport address to another.

A transport address consists of three parts: a protocol, a host address and a process association. In AS225, the protocol in a transport address is either TCP or UDP. The host address is dependent upon the protocol, but in AS225, the host address is always the internet address of the host. The process association is also protocol dependent, and in AS225, the process association is a port number. TCP/IP port numbers are 16-bit integers that are used by the transport protocols on each host to direct network traffic to the process on that host which ''owns'' that port. The set of port numbers is unique to each protocol so that port number 42 for UDP might belong to a different process than that which belongs to port number 42 for TCP. Without port numbers, multiple network programs could not run simultaneously.

Transport protocols can be thought of as analogous in some ways to Amiga devices. In such an analogy, there is a TCP and a UDP device. Each device has about 65,000 units and none of the units can be opened in shared mode.

2.3 Berkeley Socket

A "Berkeley socket" (hereafter referred to as just a "socket", but not to be confused with XNS sockets, which are really the equivalent of TCP/IP port numbers) is an endpoint of communication. Sockets are, in some ways, analogous to Exec Message Ports. Once a socket is created, it is bound to a transport address. If you can excuse the mixed-metaphors, each Exec Message Port has, as part of its name, a unit number on one of the transport protocol Devices.

Addresses can be bound to sockets in two ways. The binding can be made explicitly by the program to a specific address (as with the bind() call) or can be made indirectly to a socket-library chosen address (when the socket is first used or when it is returned from accept()). Direct binding is like giving an Exec Message Port a public name, indirect binding is like a private Exec Message Port.

Sockets are sometimes referred to as transport addresses, since each socket used by an application is always bound to exactly one transport address while in use.

The Amiga Shared Socket Library is our implementation of the standard functions for manipulating, sending data to, and receiving data from sockets. Other network APIs for TCP/IP have been written for other platforms (most notably TLI on SysVr4 systems). Programs written using sockets on one machine can communicate just fine with programs written using another API (i.e., TLI) on another machine. Sockets are not specific to TCP/IP. They can be used with different "domains". TCP/IP is one domain, another network protocol is another domain and local Inter-Process Communication (IPC) is yet another. Our socket library currently supports only TCP/IP.

Sockets are not part of TCP/IP or of any other network protocol stack--they are just an interface to a protocol stack (or to a local IPC mechanism).

2.4 How Data Moves Between Sockets

The data sent between sockets can be sent in very different ways, depending on the protocol used.

Some protocols (i.e., TCP) are connection-oriented, requiring that two sockets be connected. These protocols transmit data as a stream with no boundaries.

Other protocols (i.e., UDP) send a packet of data from one socket to another without any connection between the two. Most of these protocols make no guarantees as to whether or not the data will get there. Since a datagram is a packet of data which is sent across such a protocol, these unreliable protocols are often referred to as datagram protocols.

The details of all the various protocols and how they behave are quite complex and are beyond the scope of this paper. We are only concerned with the difference between connection-oriented stream protocols (sockets which are obtained as type SOCK_STREAM) and connectionless datagram protocols (type SOCK_DGRAM).

Many Amiga developers are used to Exec ports and messages, and desire a similar network interface. Unfortunately, such an interface is not suitable to meet all needs and TCP/IP does not even include a common protocol for the reliable delivery of messages. Such a protocol can be created on top of UDP or simulated on top of TCP.

3 Applications

Network applications require multiple programs working together. The most common architecture and mechanics for these interactions is described in this section.

3.1 Clients and Servers

Most network applications are built around a client/server model, though other schemes do exist. The client and the server generally run on different machines. A client program contacts a server program which then provides some service to the client.

A service can be as simple as echoing back text sent to the server or as complex as providing a remote login facility. The client does whatever is necessary to communicate with the server and provides the user-interface for the application.

On most networks, each machine is capable of running both client and server programs simultaneously, but on some networks each *machine* is either a client or a server and may only run programs of that type. The focus of this paper is peer-to-peer networks (the former), not client-server networks (the latter).

3.2 Application Protocols

Every program must agree on how to send data across the network and on what meaning to attach to said data. Hence, the application itself has a protocol, one which specifies how information will be passed between the client and server and what that data means. That's why there was an application-layer in the diagrams of section 2.1.

The application-specific protocol can include such things as what network protocol will be used, what initialization takes place, how any security is implemented, what format data will be in, etc.

For standard Internet applications, the application protocols are specified in detail in one or more RFCs. RFCs are the specifications for Internet protocols and standard applications. Even if you aren't implementing a standard Internet application, the RFCs offer insight into the complexities of application protocols and how they should be specified. See the **References** section of this article for more information on RFCs and standard Internet applications.

Application specifications are generally such that the applications can be ported to any machine which supports the network protocol and neither the client nor the server knows (or cares) what type of machine is at the other end. Sometimes only one half of the application (client or server) is available for a given machine. For example, the currently released version of AS225 includes an NFS client program, but no NFS server program. To use NFS with AS225 requires access to any machine with an NFS server -- it does not require any particular type of machine.

Some application protocols are more difficult to implement on some machines than on others (i.e., NFS servers are much easier to implement on a machine with Unix filesystems than on some other machines). Applications developers should avoid this situation when they can.

3.3 Kinds of Servers

There are two kinds of servers. Those which process one request at a time are called ''iterative servers''. Those which simultaneously service multiple requests (often by spawning a process to handle each request) are ''concurrent servers''. Iterative servers are generally easier to write, but are only suitable for services which can be handled quickly and/or will not be accessed by multiple clients. Applications which use connectionless protocols (UDP) frequently have iterative servers, while applications with connection-oriented protocols (TCP) usually have concurrent servers.

3.4 Starting Servers

With the AS225 package, server programs can be started in two ways. First, the program can simply be run (as from *s:user-startup*) and kept running all the time. The second is called *inetd. Inetd* is kept running all the time and it invokes any server programs only as needed. *Inetd* makes writing servers a little easier and allows a system with limited resources to still provide a wide range of services. Details on how third parties can write servers to be invoked by *inetd* are not available at the time of this writing, but will hopefully be included on an additional sheet or on the DevCon disks.

3.5 Finding Servers--All About Port Numbers

Clients ''find'' servers by address. When servers are started, they open a socket for the protocol used by the applications and bind() it to a well-known port number. A well-known port number is one agreed on in advance by the client and the server.

The client usually starts with a hostname for the server which is obtained from the user. The client uses the well-known port number, the protocol used by the application and the hostname to construct a address it can use to contact the server.

Client programs don't generally need to open any well-known port since the server can reply without knowing the client's address (with a connection-oriented protocol) or can get the address as part of receiving a datagram (with a connectionless protocol). Hence, clients usually just open a socket and start sending.

If the client doesn't explicitly bind an address to its socket before using it, the socket library assigns an arbitrary, non-conflicting port number to the socket. These port numbers are called ''ephemeral port numbers'' because they are different each time the client is run, as opposed to ''well-known port numbers'' which are always the same. Ephemeral port numbers are similar to private Exec Message Ports in that they are generally used for replying to data sent from a socket attached to that port.

3.5.1 How Well Known Port Numbers Get That Way

There are two ways for the server's well-known port to become well-known:

1) Applications can be written with the server's well-known port number hard-coded into both the client and server. This is recommended for prototyping new programs, but is a Very Bad Thing for programs which will be distributed (sample code that I write is exempted). The port number is chosen arbitrarily, but must not be one of the reserved ports and must not conflict with a port number already in use.

2) Port numbers can be configurable, and should be for production code. In programs written for AS225, the preferred method of port number configuration is the *inet:db/services* file. Write your application to call getservbyname() to get the well-known port number for your server. Configure your application by adding an entry to the *inet:db/services* file on every machine which will use the application. Many of the standard Internet applications and Unix remote services are already in *inet:db/services.* If your application isn't already included, your installation scripts should add the entry for your application to *inet:db/services.* Offer a default value, but let your user actually pick the number since your port number must not conflict with another (pre-existing) port number. Note too, the information about reserved port numbers below.

3.5.2 Reserved Ports

Port numbers 1-255 are reserved for standard Internet applications and that port numbers 256-1023 are reserved for standard Unix remote services (which are often available for machines other than Unix). You should never choose any of these port numbers for your application unless it is an implementation of a standard for which the port number is reserved. For more information on port numbering and reserved ports, see the references at the end of this paper.

4 The Shared Socket Library

The ''background'' entry of the Shared Socket Library Autodocs offers quite a bit of information on the library, and you should probably take a peek at it now.

As one of the items in the background entry explains, the primary goal of the Shared Socket Library was to provide a network API which is as compatible with standard Unix as possible. This makes porting many applications much easier, but it also creates many little quirks that cannot be ''fixed''. We argue that faithfully emulating Unix's quirks is better than creating our own, since at least you can then write more portable software and only need to remember one set of quirks. Remember this when you wish that some function returned `*void` rather than `*foo`, etc. Expect to get a few spurious compiler warnings from your nice ANSI 'C' compiler.

We haven't been able to avoid a few differences from the standard Unix functions, and we have attempted to identify all of these differences in the Autodocs, especially in the background entry.

Many functions in the library are only needed by those porting standard Unix remote services and probably should not be used by most Amiga applications. All the functions dealing with user and group ids belong in this category.

5 Programs

The DevCon disks include two example programs, *ncopy* and *ping*. *Ncopy* is a simple example of a (non-standard) network copy program which uses TCP. *Ping* is a much more complex example of writing a standard Internet application and is part of the AS255 distribution. *Ping* isn't really an application and uses a protocol we haven't even discussed (ICMP), but is a valuable example for advanced, low-level use of the library. We have not included any UDP example, but Stevens (1990) includes several excellent examples of using UDP and TCP.

Below are outlines for generic network applications using TCP and UDP.

5.1 Skeleton for Applications Using TCP

The basic outline of the core of most client/server model applications written with TCP looks something like this (the opening, initialization, and closing of the library are covered in the ''backround'' entry of the socket Autodocs):

> Server gets a socket with socket(), gets a port number with getservbyname(), builds a sockaddr_in structure (using the port number it just got) and gives it to bind() in order to attach a specific address to the socket. It then listen()s on the socket and waits to accept() any incoming connections.
>
> The client gets a server hostname from the user, gets a socket(), gets a port number with getservbyname(), builds a sockaddr_in structure and connect()s to the server's well-known address.

When the server accept()s the connection, it gets a new socket (with a different port number than the one on which it accept()ed the connection) which is connect()ed to the client. This new socket is used for all communication with the client. The server may serve one client at a time (an iterative server, as in *ncopy's server1.c*), or may give the connected socket (the one returned by accept()) to a new process (as in *ncopy's server2.c*) or may simultaneously serve multiple clients while continuing to accept() new connections (as in *ncopy's server3.c*).

Communication between the client and server is via send() and recv(). The data is a continuous stream, with any meaning assigned by the application protocol (not to be confused with the network protocol). The data is always received intact, if at all, and is always received in the absence of serious network or host machine problems. When the communications are finished, both sides s_close() the sockets which were connected.

5.2 Skeleton For Applications Using UDP

The basic outline of most client/server model applications written with UDP look something like this:

> Server gets a socket with socket(), gets a port number with getservbyname(), builds a sockaddr_in structure (using the port number it just got) and gives it to bind() in order to attach a specific address to the socket. It then loops, waiting to recvfrom() any incoming datagrams and responding to any requests in those datagrams.
>
> The client gets a server hostname from the user, gets a socket(), gets a port number with getservbyname(), builds a sockaddr_in structure and sendto()s a datagram to the server's well-known address. It then select()s to wait to recvfrom() a datagram in reply from the server or to time out because no reply has been received from the server. On time out, the client might try to re-send the datagram since it may have been lost or corrupted. Datagrams can be also be received in an order different from that in which they were sent and can be received in duplicate.

5.3 Which Protocol Is Right For My Application?

Generally, if your application requires moving bulk data, you should be using TCP. For many other applications, TCP is appropriate just because its reliability makes it easy to use. TCP is so easy to use because it provides so much functionality. The price paid for ease of use is performance.

TCP does a good job of moving bulk data from one side of the planet to another. For data which will only be sent across one physical network (one LAN), or for data sent in small pieces, TCP doesn't perform so well. A much more specific set of functionality can always provide better performance than the most general set can. Hence, for performance-critical applications which don't move bulk data, UDP is usually the protocol of choice. Unfortunately, since UDP doesn't provide reliability, the application protocol must. This means a much more complicated application protocol. It isn't nearly as bad as it sounds, though, and Stevens (1990) offers two examples.

If you are still in doubt as to which protocol your application should use, you might look at standard Internet applications for a guide. As examples, *Telnet*, SMTP (Simple Mail Transfer Protocol) and FTP (File Transfer Protocol) all use TCP, while NFS uses UDP.

6 Conclusion

This paper, the examples, the includes and the Autodocs should be enough to get you started. Writing network applications can be very complex and difficult, but is well worth the effort. This paper was intended only to introduce you to writing network applications for the Amiga with AS225, and has left a lot unsaid about the socket interface and about networking generally. In addition to the Shared Socket Library includes and Autodocs, the following books and articles are all highly recommended. Several should be required reading for anyone seriously developing any Amiga and/or Unix network applications with TCP/IP:

References

Comer, D.E. (1991a), *Internetworking with TCP/IP, Volume I, 2d: Principles, Protocols, and Architecture.* Prentice-Hall, ISBN 0-13-468505-9

> If you want more detail on how the protocols work (especially how they support internetworking), this is the place to look. Little programming information is in this volume.

Comer, D.E. and Stevens, D.L. (1991b), *Internetworking with TCP/IP, Volume II: Design, Implementation and Internals.* Prentice-Hall, ISBN 0-13-472242-6.

> There is more detail here than most application developers will want or need, but some subjects (i.e. Out-Of-Band data) are covered here better than in any other text. The text includes a complete TCP/IP implementation for Xinu. It which should be easily understandable by Amiga developers, in part because Xinu happens to have a rather Exec-like IPC mechanism.
>
> Volume I of Comer is assumed, but the information in Stevens might be sufficient.

Hunt, M. (1990), ''Commodore TCP/IP'' in *Proceedings of the 1990 North American Amiga Developer's Conference.* CATS.

> An introduction to AS225, the Amiga OS TCP/IP package.

Jesup, R. (1991), [To-be-titled], in *Proceedings of the 1991 Amiga Developer's Conferences (North American and European).* CATS.

> An introduction to the SANA-II Network Device Driver Specification. This document is

directly useful only to network hardware and network protocol developers, not to application developers.

Stevens, W.R. (1990), *Unix Network Programming*. Prentice-Hall, ISBN 0-13-949876-1.

This book starts at the beginning and methodically leads the reader through many advanced topics. If it weren't for the fact that it serves as a reference *and* a tutorial, it could be thought of as the RKMs for AS225 software development. It introduces network protocols with some detail, and sockets with great detail. It includes source and discussion of several real-world examples: *ping*, *tftp*, *rlogin*, *lpr*, *rcmd*, *rmt*, etc. Everyone in the Amiga Networking group owns a copy.

RFCs

All Internet standards start life as Requests For Comments. They are still called RFCs even if they become required, recommended, or elective. If you wish to implement a standard Internet application, you should obtain any currently applicable RFC(s) and study them closely. Here are two ways to obtain RFCs:

NIC:

Many RFCs are available online. Paper copies of all RFCs are available from the NIC, either individually or on a subscription basis (for more information contact NIC@NIC.DDN.MIL). Online copies are available via FTP or Kermit from NIC.DDN.MIL as RFC:RFC####.TXT or RFC:RFC####.PS (#### is the RFC number without leading zeroes).

Additionally, RFCs may be requested through electronic mail from the automated NIC mail server by sending a message to SERVICE@NIC.DDN.MIL with a subject line of ''RFC ####'' for text versions or a subject line of ''RFC ####.PS'' for PostScript versions. To obtain the RFC index, the subject line of your message should read ''RFC index''.

CSNET:

CSNET Coordination and Information Center (CIC)
Hotline: (617) 873-2777
10 Moulton Street, Cambridge, MA 02138
Email: cic@sh.cs.net
Info-Server requests to: info-server@sh.cs.net

The CSNET Info-Server stocks all RFCs with numbers higher than 900, unless (like RFC 900) they have been obsoleted by later RFCs. The Info-Server also stocks selected RFCs with numbers lower than 900.

The CSNET Info-Server is an automatic program that delivers information by electronic mail. To order a document from the Info-Server, send a message to ''info-server@sh.cs.net''. You do not need a subject field. The text of your message must be in a special format, such as:

```
REQUEST: rfc
 TOPIC: heLP
 Topic: RFC822
request: END
```

The text may any combination of upper-case and lower-case letters.

This request asks for two documents ''HELP'' and ''RFC822'' from the collection ''RFC''. Your message must have a ''REQUEST'' line, and one or more ''TOPIC:'' lines to specify one or more documents. The optional statement ''REQUEST: END'' terminates your specification. Any subsequent text in the message is ignored by the Info-Server.

NOTICE: The Topic: field must be of the form ''rfc822'', and NOT ''822'' or ''rfc822.txt''.

TABLE OF CONTENTS

background
socket.library/accept
socket.library/bind
socket.library/cleanup_sockets
socket.library/connect
socket.library/endhostent
socket.library/endnetent
socket.library/endprotoent
socket.library/endpwent
socket.library/endservent
socket.library/FD_SET
socket.library/get_tz
socket.library/getdomainname
socket.library/getgid
socket.library/getgroups
socket.library/gethostbyaddr
socket.library/gethostbyname
socket.library/gethostent
socket.library/gethostname
socket.library/getlogin
socket.library/getnetbyaddr
socket.library/getnetbyname
socket.library/getnetent
socket.library/getpeername
socket.library/getprotobyname
socket.library/getprotobynumber
socket.library/getprotoent
socket.library/getpwent
socket.library/getpwnam
socket.library/getpwuid
socket.library/getservbyname
socket.library/getservbyport
socket.library/getservent
socket.library/getsockname
socket.library/getsockopt
socket.library/getuid
socket.library/getumask
socket.library/inet_addr
socket.library/inet_lnaof
socket.library/inet_makeaddr
socket.library/inet_netof
socket.library/inet_network
socket.library/inet_ntoa
socket.library/listen
socket.library/rcmd
socket.library/reconfig
socket.library/recv
socket.library/s_close
socket.library/s_getsignal
socket.library/s_ioctl
socket.library/select
socket.library/selectwait
socket.library/send
socket.library/sethostent
socket.library/setnetent
socket.library/setprotoent
socket.library/setpwent
socket.library/setservent
socket.library/setsockopt
socket.library/setup_sockets
socket.library/shutdown
socket.library/socket
socket.library/strerror
socket.library/syslog
socket.library/umask

background   background

What is a socket library?

The standard programmer's interface to network protocols on most Un*x machines is the Berkley socket abstraction. It is usually provided as a link library. You should have as documentation the books "Unix Network Programming" by W. Richard Stephens (Prentice-Hall, 1990) and "Internetworking with TCP/IP, Volume II" by Douglas Comer and David Stephens (Prentice-Hall, 1991). We don't get a kick-back from Prentice-Hall, but we do use these books every day and we do know that writing programs using TCP/IP and sockets can be difficult.

Why a shared socket library?

A shared library provides many benefits. First, it greatly reduces code size. Second, it is compiler-independent. However, the most important benefit is that it is easily upgradable. New libraries with bug fixes, speed improvements, or additional functions can be utilized by existing code without recompilation. In the case of the socket library, this means we can later add support for name resolution, better configuration, DES, etc.

Who should use this?

Everyone. The link socket library that received very limited alpha distribution is obsolete. Our primary goal in writing this socket library is compatibility. BSD, SVR4, X/Open, POSIX and OSF sources have been consulted when necessary. All standard Unix socket functions, with all their peculiarities have been faithfully ported :^) Unfortunately, due to the fact that these functions were not designed with shared libraries or the Amiga in mind, some compromises were made.

How to use this?

See "What is a socket library?" above. To port existing socket code (from Un*x or from our obsolete link library), you must:

```
        OpenLibrary the socket.library and call setup_sockets()
        not call read() or write() on sockets (use send() and recv())
        call s_close() rather than close() for sockets
        call s_ioctl() rather than ioctl() for sockets
```

In case you're wondering, our old link library was compiler-specific and included the compiler's library code for read(), write(), close() and ioctl(). On Un*x machines, these functions are just calls to the kernel which can deal with sockets and other files. Obviously, this is not possible with a shared library on a machine where these functions come from link libraries, hence the s_*() functions.

About errno:

'C' programmers should be familiar with the global variable 'errno.' It is used extensively in standard socket implementations to provide details about error conditions. We take care of this in the shared library by passing a pointer to 'errno' into the shared library with setup_sockets(). You can, of course, pass a pointer to any longword-aligned, four-byte chunk of memory you own, so this will work for non-'C'-language programmers.

About integers:

All integers are assumed to be 32-bit. None are specified as long in order to maintain Un*x compatibility. If you are using a compiler which doesn't use 32-bit ints, you must make sure that all ints are converted to longs by your code.

About assembly langauge:

To be complete, we probably should include assembly header files. We don't.

About the get*() functions:

This is standard with the Un*x functions, too, but is worth noting: These function return a pointer to a static buffer. The buffer returned by a call to getX*() is cleared on the next invocation of getX*(). For example, the buffer pointed to by the return of gethostent() is cleared by a call to gethostent(), gethostbyaddr() or gethostbyname(), but not by a call to getprotoent(), etc.

As noted in the autodocs, none of the get*ent(), set*ent() or end*ent() functions should normally be used except for in porting existing programs (and internally).

About library bases:

The shared socket library returns a different library base for each OpenLib() and uses these different library bases to keep track of some global data for each opener. If you start a new process with a new context, the new process must open and initialize socket.library. Mere tasks should not access the socket.library, only processes should.

Example:

```
/* your includes */
#include <exec/types.h>
#include <exec/libraries.h>
#include <exec/execbase.h>
#include <dos/dos.h>
#include <proto/all.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <errno.h>
/* the next two lines must always be here */
#include <ss/socket.h>
struct Library *SockBase ;

/* this is the maximum number of sockets that you want */
#define MAXSOCKS 10

main()
{
      if((SockBase = OpenLibrary( "inet:libs/socket.library", 1L )) == NULL) {
            printf("Error opening socket.library\n");
            Exit(10);
      }
      setup_sockets( MAXSOCKS, &errno );

      /* normal socket code... (see AS225 dev disk for complete examples) */

      cleanup_sockets();
      CloseLibrary( SockBase ) ;
}
```

Legalese:

This shared socket library (and it's documentation) are Copyright 1991 Commodore-Amiga, Inc. All Rights Reserved. The shared socket library was written by Martin Hunt. Dale Larson helped with the documentation and testing.

Parts of this shared socket library and the associated header files are derived from code which is:

Copyright (c) 1983 Regents of the University of California.
All rights reserved.

Redistribution and use in source and binary forms are permitted provided that the above copyright notice and this paragraph are duplicated in all such forms and that any documentation, advertising materials, and other materials related to such distribution and use acknowledge that the software was developed by the University of California, Berkeley. The name of the University may not be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

```
socket.library/accept                                     socket.library/accept

   NAME
       accept -- Accept new connection from socket.

   SYNOPSIS
       ns = accept( s, name, len )
       D0           D0 A0    A1

       int accept(int, struct sockaddr *, int *);

   FUNCTION
       Servers using TCP (or another connection-oriented protocol) accept()
       connections initiated by a client program.  The connections are
       generally accept()ed on a socket which on which bind() and listen()
       have been executed.  Unless there is an error, accept() will return
       a new socket which is connected to the client.  The server can then
       use the new socket ('ns') to communicate with the client (via send()
       and recv()) and still accept() new connections on the old socket
       ('s').

       'Len' should be initialized to the amount of space pointed
       to by 'name.'  The actual size of 'name' will be returned in
       'namelen' and 'name' will contain the name of the socket initiating
       the connect().  This saves the server from needing to do a
       getpeername() for new connections.

       Accept() generally blocks until a client attempts a connect().
       You may use select() to determine whether a connection is pending,
       or use a non-blocking socket (see setsockopts()).

   INPUTS
       s            - socket descriptor.
       name         - name of the peer socket.
       len          - pointer to the length of the sockaddr struct.
                      The value returned will be the actual size of the
                      sockaddr struct that was returned.

   RESULT
       ns           - new socket descriptor or -1 (errno will be set to
                      reflect exact error condition).

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       socket(), bind(), listen()
```

```
socket.library/bind                                         socket.library/bind

   NAME
       bind -- Bind a name to a socket.

   SYNOPSIS
       return = bind( s, name, namelen )
       D0              D0 A1    D1

       int bind(int, struct sockaddr *, int);

   FUNCTION
       Assigns a name to an unnamed socket.

       Connection-oriented servers generally bind() a socket to a well-
       known address and listen() at that socket, accept()ing connections
       from clients who know the address.

       Connectionless servers generally bind() a socket to a well-known
       address and recvfrom() that socket.

       Most servers should build 'name' from a well-known port obtained
       from getservbyname().  Hard-coded ports are often used in prototype
       servers, but should never be used in production code.  For more
       information on port numbering, see your favorite TCP/IP refrence.

   INPUTS
       s               - socket descriptor.
       name            - address to bind to socket 's.'
       namelen              - length (in bytes) of 'name.'

   RESULT
       return          - 0 if successful else -1 (errno will contain the
                         specific error).

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       socket(), listen(), accept()
```

```
socket.library/cleanup_sockets                   socket.library/cleanup_sockets

   NAME
       cleanup_sockets -- Free global data for sockets.

   SYNOPSIS
       cleanup_sockets( );

       void cleanup_sockets( void );

   FUNCTION
       This function frees all signals and allocated memory.
       It closes all open sockets.  This function should be
       called after all socket functions are completed and before
       CloseLibrary().

   INPUTS
       None.

   RESULT
       None.

   BUGS

   SEE ALSO
       setup_sockets()
```

```
socket.library/connect                                   socket.library/connect

   NAME
       connect -- Connect to socket.

   SYNOPSIS
       return = connect( s, name, namelen )
       D0                D0  A1     D1

       int connect(int, struct sockaddr *, int);

   FUNCTION
       To communicate with a server using a connection-oriented protocol
       (i.e. TCP), the client must connect() a socket obtained by the
       client (from socket()) to the server's well-known address.  (The
       server will receive (from accept()) a new socket which is connected
       to the socket which the client is connect()ing to the server.)

       Clients of a connectionless server (i.e. one using UDP) can use
       connect() and send() rather than always using sendto().  In this
       case, no actual connection is created, the address to send to is
       just stored with the socket.

       Most clients should build 'name' from a well-known port obtained from
       getservbyname().  Hard-coded ports are often used in prototype
       clients, but should never be used in production code.  For more
       information on port numbering, see your favorite TCP/IP refrence.

   INPUTS
       s            - socket descriptor.
       name         - address of socket to connect 's' to.
       namelen             - length of 'name.'

   RESULT
       return       - 0 if successful else -1 (errno will contain the
                      specific error).

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       socket(), bind(), listen(), accept()
```

```
socket.library/endhostent                             socket.library/endhostent

   NAME
       endhostent -- Closes the hosts file ("inet:s/hosts").

   SYNOPSIS
       endhostent()

       void endhostent( void );

   FUNCTION
       Closes the host file if it is open.  Only needed if sethostent()
       or gethostent() have been called.

   INPUTS
       None.

   RESULT
       None.

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       gethostent(), gethostbyname(), gethostbyaddr(), sethostent()
```

```
socket.library/endnetent                           socket.library/endnetent

   NAME
       endnetent -- Closes the networks file.

   SYNOPSIS
       endnetent( )

       void endnetent( void );

   FUNCTION
       Closes the host file if it is open.  This is only needed if
       setnetent() or getnetent() has been called.

   INPUTS
       None.

   RESULT
       None.

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       getnetent(), getnetbyname(), getnetbyaddr(), setnetent()
```

```
socket.library/endprotoent                       socket.library/endprotoent

   NAME
       endprotoent -- Closes the protocols file.

   SYNOPSIS
       endprotoent( )

       void endprotoent( void );

   FUNCTION
       Closes the protocols file if it is open.  This function is needed
       only if getprotoent() or setprotoent() have been called.

   INPUTS
       None.

   RESULT
       None.

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       getprotoent(), setprotoent(), getprotobyname(), getprotobynumber()
```

```
socket.library/endpwent                                socket.library/endpwent

   NAME
       endpwent - Closes the password file.

   SYNOPSIS
       endpwent()

       void endpwent( void );

   FUNCTION
       Closes the password file if it was open.

   INPUTS
       None.

   RESULT
       None.

   SEE ALSO
       getpwent(), setpwent()
```

```
socket.library/endservent                              socket.library/endservent

   NAME
       endservent -- Closes the services file.

   SYNOPSIS
       endservent()

       void endservent( void );

   FUNCTION
       Closes the services file if it is open.  This function is needed
       only if setservent() or getservent() have been called.

   INPUTS
       None.

   RESULT
       None.

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       setservent(), getservent()
```

```
socket.library/FD_SET                                   socket.library/FD_SET

    NAME
        FD_SET -- Macros to manipulate socket masks.

    SYNOPSIS
        #include <sys/types.h>

        FD_SET(socket, mask)
        FD_CLR(socket, mask)
        result = FD_ISSET(socket, mask)
        FD_ZERO(mask)

    FUNCTION
        Type fd_set contains masks for use with sockets and select() just
        like longs can contain masks for use with signals and Wait().
        Unlike signal masks, you can't manipulate socket masks with boolean
        operators, instead you must use the FD_*() macros.

        FD_SET() sets the "bit" for 'socket' in 'mask.'

        FD_CLR() clears the "bit" for 'socket' in 'mask.'

        FD_ISSET() returns non-zero if the "bit" for 'socket' in 'mask' is
        set, else zero.

        FD_ZERO() clears all "bits" in 'mask.'  FD_ZERO should be used
        to initialize an fd_set variable before it is used.

    EXAMPLE

    SEE ALSO
        select(), selectwait()
```

```
socket.library/get_tz                                   socket.library/get_tz

    NAME
        get_tz -- Get timezone offset.

    SYNOPSIS
        offset = get_tz()
        D0

        short get_tz(void);

    FUNCTION
        Returns the number offset from UTC (Universal Time Coordinated,
        formerly Greenwich Mean Time) in minutes west of UTC.

    RESULT
        offset        - offset in minutes or -1.  An error requester
                        is displayed on error.

    NOTES
        THIS IS AN AMIGA-ONLY FUNCTION.

        Currently the configuration information is stored in memory
        in a configuration structure.  If this structure cannot be
        found, it will be initialized by reading in inet:s/inet.config.
        This may change in the future.

    BUGS
        Does not account for daylight savings time.  If the time is really
        important to you (or hosts you communicate with) you must currently
        change your inet:s/inet.config for daylight savings/standard time.
```

```
socket.library/getdomainname                    socket.library/getdomainname

  NAME
      getdomainname -- Get domain name.

  SYNOPSIS
      return = getdomainname( name, namelen)
      D0                      A1    D1

      int getdomainname (char *, int);

  FUNCTION
      Returns the name of the domain into the pointer specified.
      Name will be null-terminated if sufficient space is available.
      To find out what a domain name is, check your favorite TCP/IP
      reference.

  INPUTS
      name          - pointer to character buffer.
      namelen              - space available in 'name.'

  RESULT
      0 is returned if successful.  -1 is returned on error.

  EXAMPLE

  NOTES
      There is currently no corresponding setdomainname() function.

      Currently the configuration information is stored in memory
      in a configuration structure.  If this structure cannot be
      found, it will be initialized by reading in inet:s/inet.config.
      This may change in the future.

  BUGS

  SEE ALSO
```

```
socket.library/getgid                                    socket.library/getgid

  NAME
      getgid -- Get group id.

  SYNOPSIS
      #include <sys/types.h>

      gid = getgid()

      gid_t getgid (void);

  FUNCTION
      returns the user's group id

  INPUTS
      None.

  RESULT
      gid             - group ID or -1 (on error). An error requester
                        will be displayed if there is a problem reading
                        the current configuration.

  EXAMPLE

  NOTES
      This is an emulation of the Unix getgid() function.
      getegid() is equivalent to getgid() on the Amiga.  Note that the
      user has one primary group ID, but may have several additional
      secondary group IDs which can only be obtained with getgroups().

      Currently, the configuration information is stored in memory
      in a configuration structure.  If this structure cannot be
      found, it will be initialized by reading in inet:s/inet.config.
      This may change in the future.

  BUGS

  SEE ALSO
      getgroups(), getuid()
```

```
socket.library/getgroups                                socket.library/getgroups

   NAME
       getgroups -- Get group access list.

   SYNOPSIS
       #include <sys/types.h>
       #include <sys/param.h>

       num = getgroups(max_gids, gids)
       D0              D0        A0

       int getgroups (int, gid_t *);

   FUNCTION
       The array "gids" is filled with the group ids that the current user
       belongs to (including the primary gid).  The list may be up to
       "max_gids" long.  The actual number of gids is returned in 'num.'

   INPUTS
       max_gids        - length of gids array.
       gids            - gid_t array.

   RESULT
       num             - the number of gids is returned if successful.
                         No errors are currently defined.

   EXAMPLE
       gid_t gids[10];
       int number_of_gids;

       number_of_gids = getgroups(10,gids);

   NOTES
       Currently, the configuration information is stored in memory
       in a configuration structure.  If this structure cannot be
       found, it will be initialized by reading in inet:s/inet.config.
       This may change in the future.

       The upper limit of groups that can be returned by getgroups() is
       NGROUP (in <sys/param.h>).

   BUGS
       This routine has had problems including the primary gid.
       These problems should be fixed, so please report if you
       see this bug.

   SEE ALSO
       getgid(), getuid()
```

```
socket.library/gethostbyaddr                        socket.library/gethostbyaddr

   NAME
       gethostbyaddr -- Get host entry from the hosts file.

   SYNOPSIS
       #include <sys/socket.h>
       #include <netdb.h>

       host = gethostbyaddr( addr, len, type )
       D0                    A0    D0   D1

       struct hostent *gethostbyaddr( char *, int, int );

   FUNCTION
       Opens the host file if necessary.  Finds the entry for 'addr'
       and returns it in a hostent structure.  In the future, this
       function will look beyond just the hosts file.

       struct hostent {
       char    *h_name;        /* official name of host */
       char    **h_aliases;    /* alias list */
       int     h_addrtype;     /* host address type */
       int     h_length;       /* length of address */
       char    **h_addr_list;/* list of addresses from name server */
       #define         h_addr h_addr_list[0]/* messed up for historical reasons */
       };

   INPUTS
       addr            - internet address, in network byte order.
                         (This is really a long.)
       len             - sizeof(addr), usually 4
       type            - usually AF_INET

   RESULT
       host            - pointer to struct hostent for protocol 'addr.'
                         NULL on EOF or failure to open protocols file.

   EXAMPLE

   NOTES
       The only type currently in use is AF_INET.

       The buffer pointed to by 'host' will be overwritten by the next
       call to gethost*().

   BUGS

   SEE ALSO
       gethostbyname()
```

```
socket.library/gethostbyname                          socket.library/gethostbyname

   NAME
       gethostbyname -- Get host entry from the hosts file.

   SYNOPSIS
       #include <sys/socket.h>
       #include <netdb.h>

       host = gethostbyname( name )
       D0                    A0

       struct hostent *gethostbyname( char * );

   FUNCTION
       Opens the host file if necessary.  Finds the entry for "name"
       and returns it in a hostent structure.  In the future, this function
       will look beyond just the hosts file.

       struct hostent {
       char   *h_name;      /* official name of host */
       char   **h_aliases;  /* alias list */
       int    h_addrtype;   /* host address type */
       int    h_length;     /* length of address */
       char   **h_addr_list;/* list of addresses from name server */
       #define        h_addr h_addr_list[0] /* messed up for historical reasons */
       };

   INPUTS

   RESULT
       host          - pointer to struct hostent for protocol 'name.'
                       NULL on EOF or failure to open protocols file.

   EXAMPLE

   NOTES
       The buffer pointed to by 'host' will be overwritten by the next
       call to gethost*().

   BUGS

   SEE ALSO
       gethostbyaddr()
```

```
socket.library/gethostent                                 socket.library/gethostent

   NAME
       gethostent -- Get host entry from the hosts file ("inet:s/hosts").

   SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>
       #include <netdb.h>

       host = gethostent()
       D0

       struct hostent *gethostent( void );

   FUNCTION
       There is normally no reason to use this function.  It is used
       internally by gethostbyname() and gethostbyaddr().  It is provided
       only for Un*x compatibility and even Un*x is phasing it out in
       favor of other methods of name resolution.

       Opens the host file if necessary.  Returns the next entry
       in the file in a hostent structure.

       struct hostent {
       char   *h_name;      /* official name of host */
       char   **h_aliases;  /* alias list */
       int    h_addrtype;   /* host address type */
       int    h_length;     /* length of address */
       char   **h_addr_list;/* list of addresses from name server */
       #define        h_addr h_addr_list[0] /* messed up for historical reasons */
       };

   INPUTS
       None.

   RESULT
       host          - pointer to struct hostent for next protocol.
                       NULL on EOF or failure to open protocols file.
   EXAMPLE

   NOTES
       The buffer pointed to by 'host' will be overwritten by the next
       call to gethost*().

   BUGS

   SEE ALSO
       sethostent(), gethostbyname(), gethostbyaddr(), endhostent()
```

```
socket.library/gethostname                          socket.library/gethostname

   NAME
       gethostname -- Get the name of your Amiga.

   SYNOPSIS
       return = gethostname( name, length )
       D0                    A0    D0

       int gethostname (char *, int);

   FUNCTION
       Copies the  null-terminated hostname to 'name'.  The hostname will
       be truncated if insufficient space is available in 'name'.

   INPUTS
       name         - pointer to character array.
       length       - size of character array.

   RESULT
       return       - 0 if successful, else -1.  The only reason for
                      failure will be if the configuration file is
                      unavailable, in which case an error requester will
                      be displayed.

   NOTES
       Currently, the configuration information is stored in memory
       in a configuration structure.  If this structure cannot be
       found, it will be initialized by reading in inet:s/inet.config.
       This may change in the future.

   BUGS

   SEE ALSO
```

```
socket.library/getlogin                                socket.library/getlogin

   NAME
       getlogin -- Get login name.

   SYNOPSIS
       name = getlogin()
       D0

       char *getlogin (void);

   FUNCTION
       Returns a pointer to the current user name.

   INPUTS
       None.

   RESULT
       name         - a pointer to the current user name or NULL to
                      indicate an error.  You can use this pointer for
                      as long as necessary, but do not write to it.

   NOTES
       Currently, the configuration information is stored in memory
       in a configuration structure.  If this structure cannot be
       found, it will be initialized by reading in inet:s/inet.config.
       This may change in the future.  There is no support for multiple
       user names on a single Amiga because the Amiga OS has no concept
       of multiple users.
```

```
socket.library/getnetbyaddr                       socket.library/getnetbyaddr

   NAME
       getnetbyaddr -- Get net entry from the networks file.

   SYNOPSIS
       #include <netdb.h>

       net = getnetbyaddr( net, type )
       D0                  D0   D1

       struct netent *getnetbyaddr( long, int );

   FUNCTION
       Opens the networks file if necessary.  Returns the entry
       with a matching address in a nettent structure.

       struct netent {
               char            *n_name;/* official name of net */
               char            **n_aliases;/* alias list */
               int             n_addrtype;/* net address type */
               unsigned long n_net;/* network # */
       };

   INPUTS
       net             - network number.
       type            - network number type, currently AF_INET.

   RESULT
       net             - netent structure if succesful, NULL on EOF on
                         failure to open networks file.

   EXAMPLE

   NOTES
       The netent structure is returned in a buffer that will be
       overwritten on the next call to getnet*().

   BUGS

   SEE ALSO
       getnetbyname()
```

```
socket.library/getnetbyname                       socket.library/getnetbyname

   NAME
       getnetbyname -- Get net entry from the networks file.

   SYNOPSIS
       #include <netdb.h>

       net = getnetbyname( name )
       D0                  A0

       struct netent *getnetbyname ( char * );

   FUNCTION
       Opens the networks file if necessary.  Returns the entry
       with a matching name in a nettent structure.

       struct netent {
               char            *n_name;/* official name of net */
               char            **n_aliases;/* alias list */
               int             n_addrtype;/* net address type */
               unsigned long n_net;/* network # */
       };

   INPUTS
       name            - network name.

   RESULT
       net             - netent structure if succesful, NULL on EOF on
                         failure to open networks file.

   EXAMPLE

   NOTES
       The netent structure is returned in a buffer that will be
       overwritten on the next call to getnet*().

   BUGS

   SEE ALSO
       getnetbyaddr()
```

```
socket.library/getnetent                                    socket.library/getnetent

   NAME
       getnetent -- Get net entry from the networks file.

   SYNOPSIS
       #include <netdb.h>

       net = getnetent( )
       D0

       struct netent *getnetent( void );

   FUNCTION
       This function should not normally be used.  It is called internally
       by getnetbyname() and getnetbyaddr().

       Opens the networks file if necessary.  Returns the next entry
       in the file in a nettent structure.

       struct netent {
               char          *n_name;/* official name of net */
               char          **n_aliases;/* alias list */
               int           n_addrtype;/* net address type */
               unsigned longn_net;/* network # */
       );

   INPUTS
       None.

   RESULT
       net            - netent structure if succesful, NULL on EOF on
                        failure to open networks file.

   EXAMPLE

   NOTES
       The netent structure is returned in a buffer that will be
       overwritten on the next call to getnet*().

   BUGS

   SEE ALSO
       setnetent(), getnetbyname(), getnetbyaddr(), endnetent()
```

```
socket.library/getpeername                                  socket.library/getpeername

   NAME
       getpeername -- Get name of connected peer.

   SYNOPSIS
       return = getpeername( s, name, namelen )
       D0                    D0 A0    A1

       int getpeername( int, struct sockaddr *, int * );

   FUNCTION
       For every connected socket, there is a socket at the other end
       of the connection.  To determine the address of the socket at the
       other end of a connection, pass getpeername() the socket on your
       end.  'Namelen' should be initialized to the amount of space pointed
       to by 'name.'  The actual size of 'name' will be returned in
       'namelen.'

   INPUTS
       s              - socket descriptor.
       name           - pointer to a struct sockaddr.
       namelen              - initialized to size of 'name.' On return this
                        contains the actual sizeof(name)

   RESULT
       return         - 0 if successful else -1 (errno will contain the
                        specific error).

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       bind(), accept(), getsockname()
```

```
socket.library/getprotobyname                    socket.library/getprotobyname

    NAME
        getprotobyname -- find a protocol entry by name

    SYNOPSIS
        #include <netdb.h>

        proto = getprotobyname( name )
        D0                      A0

        struct protoent *getprotobyname( char * );

    FUNCTION
        Opens the protocols file if necessary.  Returns the entry
        with a matching name in a protoent structure.

        struct protoent {
                char    *p_name;        /* official protocol name */
                char    **p_aliases;    /* alias list */
                int     p_proto;        /* protocol # */
        };


    INPUTS
        name          - name of prototype to return.

    RESULT
        proto         - pointer to struct protoent for protocol 'name.'
                        NULL on EOF or failure to open protocols file.

    EXAMPLE

    NOTES
        The protoent structure is returned in a buffer that will be
        overwritten on the next call to getproto*()

    BUGS

    SEE ALSO
        getprotobynumber()
```

```
socket.library/getprotobynumber                socket.library/getprotobynumber

    NAME
        getprotobynumber -- find a protocol entry by number

    SYNOPSIS
        #include <netdb.h>

        proto = getprotobynumber( proto )
        D0                        D0

        struct protoent *getprotobynumber( int );

    FUNCTION
        Opens the protocols file if necessary.  Returns the entry
        with a matching protocol number in a protoent structure.

        struct protoent {
                char    *p_name;        /* official protocol name */
                char    **p_aliases;    /* alias list */
                int     p_proto;        /* protocol # */
        };


    INPUTS
        proto         - number of prototype to return.

    RESULT
        proto         - pointer to struct protoent for protocol 'number.'
                        NULL on EOF or failure to open protocols file.

    EXAMPLE

    NOTES
        The protoent structure is returned in a buffer that will be
        overwritten on the next call to getproto*()

    BUGS

    SEE ALSO
        getprotobyname()
```

```
socket.library/getprotoent                          socket.library/getprotoent

   NAME
       getprotoent -- Get a protocol entry from the protocols file.

   SYNOPSIS
       #include <netdb.h>

       proto = getprotoent ( )

       struct protoent *getprotoent ( void );

   FUNCTION
       There is normally no reason to use this function.  It is used
       internally by getprotobyname() and getprotobynumber().  It is
       provided only for Un*x compatibility.

       Opens the protocols file if necessary.  Returns the next entry
       in the file in a protoent structure.

       struct protoent {
               char    *p_name;        /* official protocol name */
               char    **p_aliases;    /* alias list */
               int     p_proto;        /* protocol # */
       };


   INPUTS
       None.

   RESULT
       proto          - NULL on EOF or failure to open protocols file.

   EXAMPLE

   NOTES
       The protoent structure is returned in a buffer that will be
       overwritten on the next call to getproto*()

   BUGS

   SEE ALSO
       getprotobyname(), getprotobynumber(), setprotoent(), endprotoent()
```

```
socket.library/getpwent                                  socket.library/getpwent

   NAME
       getpwent -- Read the next line in the password file.

   SYNOPSIS
       passwd = getpwent()
       D0

       struct passwd *getpwent( void ) ;

   FUNCTION
       There is usually no reason to call this function directly.  It
       is called internally by getpwuid() and getpwnam().  It is provided
       only for Un*x compatibility.

       Opens the password file if necessary.  Returns the next entry
       in the file in a passwd structure.

       struct passwd {
               char   *pw_name;
               char   *pw_dir;
               char   *pw_passwd;
               char   *pw_gecos;
               uid_t  pw_uid;
               gid_t  pw_gid;
               char   *pw_shell;/* currently unused */
               char   *pw_comment;
       };

   INPUTS
       None.

   RESULT
       passwd                - a pointer to a filled in passwd structure
                    if successful, else NULL.

   SEE ALSO
       getpwuid(), getpwnam(), setpwent(), endpwent()
```

```
socket.library/getpwnam                                socket.library/getpwnam

   NAME
       getpwnam -- Search user database for a particular name.

   SYNOPSIS
       #include <pwd.h>

       passwd = getpwnam( name )
       D0                 A0

       struct passwd *getpwnam( char * );

   FUNCTION
       getpwnam() returns a pointer to a passwd structure. The passwd
       structure fields are filled in from the fields contained in
       one line of the password file.

   INPUTS
       name          - user name of passwd entry to look up.

   RESULT
       passwd              - a pointer to a passwd struct with its elements
                     filled in from the passwd file.

              struct passwd {
                     char   *pw_name;
                     char   *pw_dir;
                     char   *pw_passwd;
                     char   *pw_gecos;
                     uid_t  pw_uid;
                     gid_t  pw_gid;
                     char   *pw_shell;/* currently unused  */
                     char   *pw_comment;
              };

   NOTES
       The passwd structure is returned in a buffer that will be
       overwritten on the next call to getpw*().

   SEE ALSO
       getpwuid()
```

```
socket.library/getpwuid                                socket.library/getpwuid

   NAME
       getpwuid -- Search user database for a particular uid.

   SYNOPSIS
       #include <pwd.h>

       passwd = getpwuid( uid )
       D0                 D1

       struct passwd *getpwuid( uid_t );

   FUNCTION
       getpwuid() returns a pointer to a passwd structure.  The passwd
       structure fields are filled in from the fields contained in
       one line of the password file.

   INPUTS
       uid           - uid of passwd entry to look up.

   RESULT
       passwd              - a pointer to a passwd struct with its elements
                     filled in from the passwd file.

              struct passwd {
                     char   *pw_name;
                     char   *pw_dir;
                     char   *pw_passwd;
                     char   *pw_gecos;
                     uid_t  pw_uid;
                     gid_t  pw_gid;
                     char   *pw_shell;/* currently unused */
                     char   *pw_comment;
              };

   NOTES
       The passwd structure is returned in a buffer that will be
       overwritten on the next call to getpw*().

   SEE ALSO
       getpwnam()
```

```
socket.library/getservbyname                          socket.library/getservbyname

  NAME
     getservbyname -- Find a service entry by name.

  SYNOPSIS
     #include <netdb.h>

     serv = getservbyname( name, proto )
     D0                    A0    A1

     struct servent *getservbyname( char *, char * );

  FUNCTION
     Opens the services file and finds the service with the matching
     name and protocol.

     struct servent {
             char   *s_name;/* official service name */
             char   **s_aliases;/* alias list */
             int    s_port;/* port # */
             char   *s_proto;/* protocol to use */
     };

  INPUTS
     name           - name of service to look up.
     proto          - protocol of service to look up.

  RESULT
     serv           - pointer to struct servent for service 'name.'
                      NULL on EOF or failure to open protocols file.

  EXAMPLE

  NOTES
     The servent structure is returned in a buffer that will be
     overwritten on the next call to getserv*().

  BUGS

  SEE ALSO
     getservent(), getservbyport()
```

```
socket.library/getservbyport                          socket.library/getservbyport

  NAME
     getservbyport -- Find a service entry by port.

  SYNOPSIS
     #include <netdb.h>

     serv = getservbyport( port, proto )
     D0                    D0    A0

     struct servent *getservbyport( u_short, char * );

  FUNCTION
     Opens the services file and finds the service with the matching
     port and protocol.

     struct servent {
             char   *s_name;/* official service name */
             char   **s_aliases;/* alias list */
             int    s_port;/* port # */
             char   *s_proto;/* protocol to use */
     };

  INPUTS

  RESULT
     serv           - pointer to struct servent for service 'name.'
                      NULL on EOF or failure to open protocols file.

  EXAMPLE

  NOTES
     The servent structure is returned in a buffer that will be
     overwritten on the next call to getserv*().

  BUGS

  SEE ALSO
     getservent(), getservbyname()
```

```
socket.library/getservent                       socket.library/getservent

 NAME
    getservent -- Get a service entry from the services file.

 SYNOPSIS
    #include <netdb.h>

    serv = getservent( void )
    D0

    struct servent *getservent( void );

 FUNCTION
    There is normally no reason to use this function.  It is used
    internally by getservbyname() and getservbyport().  It is
    provided only for Un*x compatibility.

    Opens the services file if necessary.  Returns the next entry
    in the file in a servent structure.

    struct servent {
           char   *s_name;/* official service name */
           char   **s_aliases;/* alias list */
           int    s_port;/* port # */
           char   *s_proto;/* protocol to use */
    };

 INPUTS
    None.

 RESULT
    serv          - pointer to struct servent for next service.
                    NULL on EOF or failure to open protocols file.

 EXAMPLE

 NOTES
    The servent structure is returned in a buffer that will be
    overwritten on the next call to getserv*()

 BUGS

 SEE ALSO
    setservent(), getservbyname(), getservbyport(), endservent()
```

```
socket.library/getsockname                      socket.library/getsockname

 NAME
    getsockname -- Get the name of a socket.

 SYNOPSIS
    return = getsockname( s, name, namelen )
    D0                    D0 A0    A1

    int getsockname( int, struct sockaddr *, int * );

 FUNCTION
    Returns the name (address) of the specified socket.  'Namelen'
    should be initialized to the amount of space pointed to by 'name.'
    The actual size of 'name' will be returned in 'namelen.'

 INPUTS
    s              - socket descriptor.
    name           - socket name.
    namelen                - size of 'name' (in bytes).

 RESULT
    return         - 0 if successful else -1 (errno will be set to
                     one of the following error codes:
                     EBADFinvalid socket

 EXAMPLE

 NOTES

 BUGS

 SEE ALSO
    bind(), getpeername()
```

```
socket.library/getsockopt                               socket.library/getsockopt

   NAME
       getsockopt -- Get socket options.

   SYNOPSIS
       return = getsockopt( s, level, optname, optval, optlenp )
       D0                   D0 D1     D2       A0      A1

       int getsockopt( int, int, int, char *, int * );

   FUNCTION
       Gets the option specified by 'optname' for socket 's.'
       This is an advanced function.  See the "sys/socket.h" header and
       your favorite TCP/IP reference for more information on options.

   INPUTS
       s               - socket descriptor.
       level           - protocol level.  Valid levels are:
                         IPPROTO_IP                   IP options
                         IPPROTO_TCP                  TCP options
                         SOL_SOCKET                   socket options
       optname              - option name.
       optval          - pointer to the buffer which will contain the
                         answer.
       optlen          - initially sizeof(optval). reset to new value
                         on return

   RESULT
       return          - 0 if successful else -1 (errno will contain the
                         specific error).

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       setsockopt()
```

```
socket.library/getuid                                       socket.library/getuid

   NAME
       getuid  -- Get user id.

   SYNOPSIS
       #include <sys/types.h>

       uid = getuid()
       D0

       uid_t getuid (void);

   FUNCTION
       Returns the user id.

   INPUTS
       None.

   RESULT
       uid             - user ID or -1 (on error).  An error requester will
                         be displayed if there is a problem reading the
                         current configuration.

   EXAMPLE

   NOTES
       This is an emulation of the Unix getuid() function.
       geteuid() is equivalent to getuid() on the Amiga.

       Currently, the configuration information is stored in memory
       in a configuration structure.  If this structure cannot be
       found, it will be initialized by reading in inet:s/inet.config.
       This may change in the future.  There is no support for multiple
       user IDs on a single Amiga because the Amiga OS has no concept
       of multiple users.

   BUGS

   SEE ALSO
       getgid(), getgroups()
```

```
socket.library/getumask                          socket.library/getumask

  NAME
      getumask -- Get user file creation mask.

  SYNOPSIS
      #include <sys/types.h>

      umask = getumask()
      D0

      mode_t getumask (void);

  FUNCTION
      Returns the umask.

  RESULT
      -1 will be returned and an error message will be displayed
      if there is a problem reading the current configuration.

  EXAMPLE

  NOTES
      THIS IS AN AMIGA-SPECIFIC FUNCTION.  It is not a standard Unix
      function.  See umask().

      Currently, the configuration information is stored in memory
      in a configuration structure.  If this structure cannot be
      found, it will be initialized by reading in inet:s/inet.config.
      This may change in the future.

  BUGS

  SEE ALSO
      umask()
```

```
socket.library/inet_addr                                    socket.library/inet_addr

  NAME
      inet_addr -- make internet address from string

  SYNOPSIS
      #include <sys/types.h>
      #include <sys/socket.h>
      #include <netinet/in.h>

      addr = inet_addr( string )
      D0                A1

      u_long inet_addr ( char * );

  FUNCTION
      Converts a string to an internet address.  All internet addresses
      are in network order.

  INPUTS
      string       address string. "123.45.67.89" for example

  RESULT
      A long representing the network address in network byte order.

      INADDR_NONE is returned if input is invalid.

  EXAMPLE

  NOTES

  BUGS

  SEE ALSO
```

```
socket.library/inet_lnaof                              socket.library/inet_lnaof

   NAME
       inet_lnaof -- give the local network address

   SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>
       #include <netinet/in.h>

       addr = inet_lnaof( in )
       D0                 D1

       int inet_lnaof ( struct in_addr );

   FUNCTION
       Returns the local network address.

   INPUTS
       in           - struct in_addr to find local network part of.

       struct in_addr {
              u_long s_addr; /* a long containing the internet address */
       };

   RESULT
       addr         -  the local network address portion of 'in.'

   EXAMPLE

   NOTES
       'in' is *NOT* a pointer to a struct in_addr, it IS a structure.
       It is a 4-byte structure and is a passed as a structure (rather
       than a long or pointer to a structure) for historical reasons.

   RESULT

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       inet_addr(), inet_makeaddr(), inet_netof()
```

```
socket.library/inet_makeaddr                        socket.library/inet_makeaddr

   NAME
       inet_makeaddr -- make internet address from network and host

   SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>
       #include <netinet/in.h>

       addr = inet_makeaddr( net, lna )
       D0                    D0   D1

       struct in_addr inet_makeaddr( int, int );

   FUNCTION
       Formulate an Internet address from network + host.  Used in
       building addresses stored in the ifnet structure.

       'in' is *NOT* a pointer to a struct in_addr, it IS a structure.
       It is a 4-byte structure and is a passed as a structure (rather
       than a long or pointer to a structure) for historical reasons.
       See NOTES.


   INPUTS
       net           - network number in local integer format.
       lna           - local node address in local integer format.

   RESULT
       in            - struct in_addr.

       struct in_addr {
              u_long s_addr; /* a long containing the internet address */
       };

   EXAMPLE

   NOTES
       'in' is *NOT* a pointer to a struct in_addr, it IS a structure.
       It is a 4-byte structure and is a passed as a structure (rather
       than a long or pointer to a structure) for historical reasons.

   BUGS

   SEE ALSO
       inet_addr(), inet_lnaof(), inet_netof()
```

```
socket.library/inet_netof                              socket.library/inet_netof

   NAME
       inet_netof -- give the network number of an address

   SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>
       #include <netinet/in.h>

       net = inet_netof( in )
       D0                D1

       int inet_netof( struct in_addr );

   FUNCTION
       Returns the network number.

   INPUTS
       in              - struct in_addr to find network part of.

       struct in_addr {
               u_long s_addr; /* a long containing the internet address */
       };

   RESULT
       net             - network part of 'in.'
   EXAMPLE

   NOTES
       'in' is *NOT* a pointer to a struct in_addr, it IS a structure.
       It is a 4-byte structure and is a passed as a structure (rather
       than a long or pointer to a structure) for historical reasons.

   BUGS

   SEE ALSO
       inet_addr(), inet_makeaddr(), inet_lnaof()
```

```
socket.library/inet_network                          socket.library/inet_network

   NAME
       inet_network -- make network number from string

   SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>
       #include <netinet/in.h>

       net = inet_network( string )
       D0                  A1

       int inet_network( char * );

   FUNCTION
       Converts a string to an network number if the string contains the
       dotted-decimal representation of a network number. All network
       numbers are in network order.

   INPUTS
       string          - network string. "123.45.67.89" for example.

   RESULT
       net             - INADDR_NONE is returned if input is invalid, else
                         the network number corresponding to 'string.'

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
```

```
socket.library/inet_ntoa                              socket.library/inet_ntoa

  NAME
      inet_ntoa -- turn internet address into string

  SYNOPSIS
      #include <sys/types.h>
      #include <sys/socket.h>
      #include <netinet/in.h>

      string = inet_ntoa( in )
      D0                  D1

      char *inet_ntoa ( struct in_addr );

  FUNCTION
      Converts an internet address to an ASCII string in the format
      "a.b.c.d" (dotted-decimal notation).

  INPUTS
      in           - struct in_addr.

      struct in_addr {
            u_long s_addr; /* a long containing the internet address */
      };

  RESULT
      Pointer to a string containing the ASCII ethernet address.
      For example, if in.s_addr = 0xc009d203 then a pointer
      to the string "192.9.210.3" is returned.

  NOTES
      The result points to a static buffer that is overwritten
      with each call.

      'in' is *NOT* a pointer to a struct in_addr, it IS a structure.
      It is a 4-byte structure and is a passed as a structure (rather
      than a long or pointer to a structure) for historical reasons.

  BUGS

  SEE ALSO
```

```
socket.library/listen                                    socket.library/listen

  NAME
      listen -- Indicate willingness to accept() connections.

  SYNOPSIS
      return = listen( s, backlog )
      D0               D0 D1

      int listen(int, int);

  FUNCTION
      This function is used for a connection-oriented server (i.e. one
      using the TCP protocol)to indicate that it is waiting to receive
      connections.  It is usually executed after socket() and bind(), and
      before accept().

  INPUTS
      s              - socket descriptor.
      backlog              - max number of connection requests to queue
                       (usually 5).

  RESULT
      return         - 0 is returned if successful, else  -1. On error,
                       errno will be set to one of the following:

      EBADF          - s is not a valid descriptor
      ENOTSOCK       - s is not a socket
      ECONNREFUSED   - connection refused, normally because the queue
                       is full
      EOPNOTSUPP     - s is a socket type which does not support this
                       operation.  s must be type SOCK_STREAM.

  BUGS
      'backlog' is currently limited to 5.

  SEE ALSO
      accept(), bind(), connect(), socket()
```

```
socket.library/rcmd                                    socket.library/rcmd

  NAME
      rcmd - Allow superuser to execute commands on remote machines

  SYNOPSIS
      rem = rcmd( ahost, inport, luser, ruser, cmd, fd2p )
      D0          D0     D1      A0     A1     A2   D2

      int rcmd( char **, u_short, char *, char *, char *, int *);

  FUNCTION
      This function is used by rsh and rcp to communicate with rshd.

      The rcmd subroutine looks up the host *ahost using gethostbyname,
      returning (-1) if the host does not exist. Otherwise *ahost is
      set to the standard name of the host and a connection is
      established to a server residing at the well-known Internet port
      inport.

      If the call succeeds, a socket of type SOCK_STREAM is returned to
      the caller and given to the remote command as stdin and stdout.
      If fd2p is nonzero, then an auxiliary channel to a control
      process will be set up, and a descriptor for it will be placed in
      *fd2p. The control process will return diagnostic output from the
      command (unit 2) on this channel, and will also accept bytes on
      this channel as being UNIX signal numbers, to be forwarded to the
      process group of the command. If fd2p is 0, then the stderr (unit
      2 of the remote command) will be made the same as the stdout and
      no provision is made for sending arbitrary signals to the remote
      process, although you may be able to get its attention by using
      out-of-band data.

  INPUTS
      ahost             - pointer to a pointer to a host name.
      inport            - an Internet port.
      luser             - the local user's name.
      ruser             - the remote user's name.
      cmd               - the command string to be executed.
      fd2p              - a flag telling whether to use an auxillary channel.

  RESULT
      rem          - socket of type SOCK_STREAM if successful, else -1.
```

```
socket.library/reconfig                              socket.library/reconfig

  NAME
      reconfig - re-initialize the internal configuration structure

  SYNOPSIS
      return = reconfig()
      D0

      BOOL reconfig( VOID ) ;

  FUNCTION
      Causes the socket library to read the inet:s/inet.config file.
      This is useful for when you have changed an entry in the file
      and need the system to recognize the change without a system
      reboot.

  INPUTS
      None

  RESULT
      Boolean return - TRUE upon success, FALSE upon error

  NOTES
      Make -no- assumptions about how this works internally. The current
      mechanism is in transition and is guaranteed to change.

  BUGS

  SEE ALSO
```

```
socket.library/recv                                         socket.library/recv

   NAME
       recv, recvfrom, recvmsg -- Receive a message from a socket.

   SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       numbytes = recv( s, buf, len, flags )
       D0               D0 A0   D1   D2

       numbytes = recvfrom( s, buf, len, flags, from, fromlen )
       D0                   D0 A0   D1   D2     A1    A2

       numbytes = recvmsg( s, msg, flags )
       D0                  D0 A0   D1

       int recv(int, char *, int, int )
       int recvfrom( int, char *, int, int, struct sockaddr *, int *)
       int recvmsg(int, struct msghdr *, int )

   FUNCTION
       recv() is used to receive messages on an already connect()ed
       socket.

       recvfrom() receives data from a socket whether it is in a connected
       state or not.

       recvmsg() is the most general of the recv calls and is for advanced
       use.

       If no data is available, these calls block unless the socket is
       set to nonblocking in which case (-1) is returned with errno set
       to EWOULDBLOCK.

   INPUTS
       s            - a socket descriptor.
       buf          - the buffer into which the incoming data will be
                      placed.
       len          - the size of the buffer.
       flags        - select options (MSG_OOB, MSG_PEEK).
       from         - a pointer to a sockaddr structure.
       fromlen      - Length of the 'from' buffer.
       msg          - pointer to a struct msghdr, defined in
                      "sys/socket.h."

   RESULT
       numbytes     - number of bytes read if successful else -1.

   NOTES
       'fromlen' is passed with the length of the 'from' buffer.  If 'from'
       is non-zero, the structure will be filled with the source address
       and fromlen will be filled in to represent the size of the actual
       address stored in 'from'.

   SEE ALSO
       send(), socket(), connect(), bind(), listen(), accept()
```

```
socket.library/s_close                                   socket.library/s_close

   NAME
       s_close -- Close a socket.

   SYNOPSIS
       status = s_close( socket ) ;
       D0                D0

       int s_close( int ) ;

   FUNCTION
       This function closes a socket.

   INPUTS
       unit          - socket number.

   RESULT
       status          - 0 if successful, else -1.

   EXAMPLE

   NOTES
       s_close() must always be used to close a socket.  This shared
       library does not know about filehandles or file descriptors.

   BUGS

   SEE ALSO
       socket()
```

```
socket.library/s_getsignal                          socket.library/s_getsignal

   NAME
      s_getsignal -- Get a network signal bit.

   SYNOPSIS
      signal = s_getsignal( type );
      D0                    D1

      BYTE s_getsignal( UWORD );

   FUNCTION
      This function returns a socket signal.  The socket signal can be
      used to Wait() on an event for the shared socket library.  The
      following signal types are supported:

      SIGIO   This signal indicates a socket is ready for
              asynchronous I/O.  This signal will be sent only if
              the socket has been set to async by calling ioctl()
              with a command of FIOASYNC.

      SIGURG  This signal indicates the presence of urgent or
              out-of-band data on a TCP socket.
   INPUTS
      type          - SIGIO or SIGURG.

   RESULT
      signal        - signal bit (0..31) or -1 if 'type' was invalid.

   EXAMPLE

   NOTES
      The SIGIO signal will only be set for sockets on which FIOASYNC has
      been set (with s_ioctl or s_setsockopts.)

   BUGS

   SEE ALSO
      s_ioctl(), select(), selectwait()
```

```
socket.library/s_ioctl                                          socket.library/s_ioctl

   NAME
      s_ioctl -- Control socket options.

   SYNOPSIS
      return = s_ioctl( s, cmd, data )
      D0                D0 D1   A0

      int s_ioctl ( int, int, char * );

   FUNCTION
      Manipulates device options for a socket.

   INPUTS
      s             - socket descriptor.
      cmd           - command.
      data          - data.

      The following commands are supported:

      command          description                      data points to
      -------          -----------                      --------------
      FIONBIO          set/clear nonblocking I/O        int
      FIOASYNC         set/clear async I/O              int
      FIONREAD         get number of bytes to read      int
      SIOCATMARK       at out-of-band mark?             int
      SIOCSPGRP        set process group                int
      SIOCGPGRP        get process group                int
      SIOCADDRT        add route                        struct rtentry
      SIOCDELRT        delete route                     struct rtentry
      SIOCGIFCONF      get ifnet list                   struct ifconf
      SIOCGIFFLAGS     get ifnet flags                  struct ifreq
      SIOCSIFFLAGS     set ifnet flags                  struct ifreq
      SIOCGIFMETRIC    get IF metric                    struct ifreq
      SIOCSIFMETRIC    set IF metric                    struct ifreq
      SIOCGARP         get ARP entry                    struct arpreq
      SIOCSARP         get ARP entry                    struct arpreq
      SIOCDARP         delete ARP entry                 struct arpreq

      For more information, see a Unix reference manual.

   RESULT
      return        - 0 on success, else -1 (errno will be set to the
                      specific error code).

   EXAMPLE
      int one = 1;
      ioctl ( s, FIOASYNC, (char *)&one);

   NOTES
      The standard Unix ioctl() function operates on both files and
      sockets.  Some compiler vendors may supply an ioctl() function.
      Because of this, and because this function works only with
      sockets, it has been renamed to s_ioctl().

   BUGS

   SEE ALSO
      setsockopts()
```

```
socket.library/select                                  socket.library/select

   NAME
      select -- Examines specified sockets' read, write or exception status.

   SYNOPSIS
      num = select( numfds, readfds, writefds, exceptfds, timeout )
      D0            D0      A0       A1        A2         D1

      int select( int, fd_set *, fd_set *, fd_set *, struct timeval *);

   FUNCTION
      select() examines the socket masks specified 'readfds,' 'writefds,'
      and 'execptfds' (see FD_SET) to see if they are ready for reading,
      for writing, or have an exceptional condition pending.

      When select() returns, the masks will have been modified so that
      only the "bits" for those sockets on which an event has occured are
      set.  The total number of ready sockets is returned in 'num.'

      If 'timeout' is a non-NULL pointer, it specifies a maximum
      interval to wait for the selection to complete. If timeout is
      a NULL pointer, the select blocks indefinitely. To affect a
      poll, the timeout argument should be nonzero, pointing to a
      zero valued timeval structure.  As you know, busy-loop polling
      is a no-no on the Amiga.

      Any of readfds, writefds, and execptfds may be given as NULL if
      any of those categories are not of interest.

      select() should not be used with a single socket (use s_getsignal()
      and Wait() instead).

   INPUTS
      numfds     - Maximum number of bits in the masks that
                   represent valid file descriptors.
      readfds    - 32 bit mask representing read file descriptors
      writefds   - 32 bit mask representing write file descriptors
      exceptfds  - 32 bit mask representing except file descriptors
      timeout    - Pointer to a timeval structure which holds the
                   maximum amount of time to wait for the selection
                   to complete.

   RESULT
      num        - The number of ready sockets, zero if a timeout occurred,
                   -1 on error.
      readfds    - A mask of the ready socket descriptors
      writefds   -     "      "    "     "       "
      exceptfds  -     "      "    "     "       "

   NOTES
      If a process is blocked on a select() waiting for input from a
      socket and the sending process closes the socket, the select
      notes this as an exception rather than as data.  Hence, if
      the select is not currently looking for exceptions, it will
      wait forever.

      The descriptor masks are always modified on return, even if
      the call returns as the result of the timeout.

      The current version of this function calls selectwait()
      with a control-C option in the umask field.

      A common error is to use the socket number in which you are
      interested as the first argument. Use socket+1.

   BUGS

   SEE ALSO
      FD_SET(), selectwait(), s_getsignal()
```

```
socket.library/selectwait                          socket.library/selectwait

   NAME
      selectwait -- select() with optional, task specific Wait() mask.

   SYNOPSIS
      num = selectwait( numfds, rfds, wfds, exfds, time, umask )
      D0                D0      A0    A1    A2     D1    D2

      int selectwait (int, int *, int *, int *, struct timeval *, long *);

   FUNCTION
      selectwait() is the same as select() except that it processes
      one additional argument, 'umask'

      The umask argument should contain either a NULL or a mask
      of the desired task-specific signal bits that will be tested
      along with the socket descriptors.  selectwait() does a standard
      Exec Wait() call and adds the supplied mask value to Wait()
      argument.

   INPUTS
      numfds     - The maximum number of bits in the masks that
                   represent valid file descriptors.
      readfds    - 32 bit mask representing read file descriptors
      writefds   - 32 bit mask representing write file descriptors
      exceptfds  - 32 bit mask representing except file descriptors
      timeout    - A pointer to a timeval structure which holds the
                   maximum amount of time to wait for the selection
                   to complete.
      umask      - A mask of the task's signal bits that will be
                   used (in addition to the standard select() call
                   return options) to have the call return. This can
                   be SIGBREAKF signals, Intuition userport signals,
                   console port signals, etc. Any mask that you would
                   pass to the the Exec Wait() call is ok here.

   RESULT
      num        - The number of ready file descriptors if
                   successful.  Zero (0) if a timeout occurred.
                   (-1) upon error.
      readfds    - A mask of the ready file descriptors
      writefds   -     "    "   "    "     "
      exceptfds  -     "    "   "    "     "
      umask      - A mask of the bits in the originally passed 'umask'
                   variable that have actually occured.

   EXAMPLE
      long umask = SIGBREAKF_CTRL_D | 1L << myport->mp_SigBit ;

      numfds = selectwait(n, r, w, e, time, &umask) ;
      if( event & SIGBREAKF_CTRL_D ) {
            printf("user hit CTRL-D\n") ;
      }
      if( event & 1L << myport->mp_SigBit ) {
            printf( "myport got a message\n" ) ;
      }
   NOTES
      A common error is to use the socket number in which you are
      interested as the first argument. Use socket+1.

   BUGS

   SEE ALSO
      FD_SET(), select(), s_getsignal()
```

```
socket.library/send                                       socket.library/send

   NAME
       send, sendto, sendmsg -- Send data from a socket.

   SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       cc = send( s, buf, len, flags )
       D0         D0 A0   D1   A1

       cc = sendto ( s, buf, len, flags, to, to_len )
       D0            D0 A0   D1   D2     A1  D3

       cc = sendmsg( s, msg, flags )
       D0            D0 A0   D1

       int send (int, char *, int, int );
       int sendto (int, char *, int, int, struct sockaddr *, int );
       int sendmsg (int, struct msghdr *, int );

   FUNCTION
       send(), sendto(), and sendmsg() transmit data from a socket.
       send() must be used with a connect()ed socket.  sendto() can
       only be used with non-connect()ed DGRAM-type sockets.  sendmsg()
       is an advanced function.

   INPUTS
       s               - socket descriptor.
       buf             - pointer to message buffer.
       len             - length of message to transmit.
       flags           - 0 or MSG_OOB to send out-of-band data.
       to              - pointer to a sockaddr containing the destination.
       to_len          - sizeof(struct sockaddr).
       msg             - pointer to a struct msghdr, defined in
                         "sys/socket.h."

   RESULT
       cc              - the number of bytes sent. This does not imply that
                         the bytes were recieved.  -1 is returned on a local
                         error (errno will be set to the specific error
                         code).  Possible errors are:

               EBADF        an invalid descriptor was used
               ENOTSOCK     's' is not a socket
               EMSGSIZE     the socket requires that the message be
                            sent atomically and the size of the message
                            prevents that.
               EWOULDBLOCK  requested operation would block

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       recv(), socket()
```

```
socket.library/sethostent                           socket.library/sethostent

   NAME
       sethostent -- Rewind the hosts file ("inet:s/hosts").

   SYNOPSIS
       sethostent( flag )

       void sethostent( int );
                        D1

   FUNCTION
       This function is rarely useful.

       Opens the host file if necessary.  Rewinds the host file if it
       is open.

   INPUTS
       flag            - If 'flag' is 1, calls to gethostbyname() and
                         gethostbyaddr() will not close the file between
                         calls.  You must close the file with an
                         endhostent().  Once set, 'flag' cannot be reset
                         except by calling endhostent().

   RESULT
       None.

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       gethostent(), gethostbyname(), gethostbyaddr(), endhostent()
```

```
socket.library/setnetent                              socket.library/setnetent

   NAME
       setnetent -- Open or rewind the networks file.

   SYNOPSIS
       setnetent( flag )
                  D1

       void setnetent( int );

   FUNCTION
       Rewinds the network file if it is open.
       Opens the network file if it was not open.

   INPUTS
       flag         - if 'flag' is 1, calls to getnetbyname() and
                      getnetbyaddr() will not close the file between
                      calls.  You must close the file with an endnetent().
                      Once set, 'flag' cannot be reset except by calling
                      endnetent().

   RESULT
       None.

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       getnetent(), getnetbyname(), getnetbyaddr(), endnetent()
```

```
socket.library/setprotoent                          socket.library/setprotoent

   NAME
       setprotoent -- Open or rewind the protocols file.

   SYNOPSIS
       setprotoent( flag )

       void setprotoent( int );
                         D1

   FUNCTION
       This function is rarely useful.

       Opens the protocols file if necessary.
       Rewinds the protocols file if it is open.

   INPUTS
       flag         - if 'flag' is 1, calls to getprotobyname() and
                      getprotobynumber() will not close the file
                      between calls.  You must close the file with an
                      endprotoent().  Once set, 'flag' cannot be reset
                      except by calling endprotoent().
   RESULT
       None.

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       getprotoent(), endprotoent(), getprotobyname(), getprotobynumber()
```

```
socket.library/setpwent                               socket.library/setpwent

   NAME
       setpwent - Opens or rewinds the password file.

   SYNOPSIS
       setpwent( flag )
                 D1

       void setpwent( int );

   FUNCTION
       If the file is already open the file is rewound. Otherwise the
       file is opened.

   INPUTS
       flag           - if 'flag' is 1, calls to getpwuid() and getpwnam()
                        will not close the file between calls.  You must
                        close the file with an endpwent().  Once set,
                        'flag' cannot be reset except by calling endpwent().

   RESULT
       None.

   SEE ALSO
       endpwent(), getpwent()
```

```
socket.library/setservent                           socket.library/setservent

   NAME
       setservent -- Open or rewind the services file.

   SYNOPSIS
       setservent( flag )
                   D1

       void setservent( int );

   FUNCTION
       This function is rarely useful.

       Opens the services file if necessary.
       Rewinds the services file if it is open.

   INPUTS
       flag           - if 'flag' is 1, calls to getservbyport() and
                        getservbyname() will not close the file between
                        calls.  You must close the file with an
                        endservent().  Once set, 'flag' cannot be reset
                        except by calling endservent().

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       getservent(), endservent()
```

```
socket.library/setsockopt                              socket.library/setsockopt

  NAME
     setsockopt -- Set socket options.

  SYNOPSIS
     return = setsockopt( s, level, optname, optval, optlen )
     D0                   D0 D1     D2       A0      D3

     int setsockopt( int, int, int, char *, int );

  FUNCTION
     Sets the option specified by 'optname' for socket 's.'
     This is an advanced function.  See the "sys/socket.h" header and
     your favorite TCP/IP reference for more information on options.

  INPUTS
     s             - socket descriptor.
     level         - protocol level.  Valid levels are:
                      IPPROTO_IPIP options
                      IPPROTO_TCPTCP options
                      SOL_SOCKETsocket options
     optname             - option name.
     optval        - pointer to the buffer with the new value.
     optlen        - size of 'optval' (in bytes).

  RESULT
     return        - 0 if successful else -1 (errno will contain the
                     specific error).

  EXAMPLE
           int on = 1;
           setsockopt( s, SOL_SOCKET, SO_DEBUG, &on, (int)sizeof(on));

  NOTES

  BUGS

  SEE ALSO
     getsockopt()
```

```
socket.library/setup_sockets                        socket.library/setup_sockets

  NAME
     setup_sockets -- Initialize global data for sockets.

  SYNOPSIS
     retval = setup_sockets( max_sockets, errnop );
     D0                      D1           A0

     ULONG setup_sockets( UWORD, int * );

  FUNCTION
     This function initializes global variables, sets the library errno
     to point to the application's errno, and allocates signals.
     This should always be called immediately after an OpenLibrary().
     The only reason this initialization is not done automatically
     is because a pointer to errno must be passed in.

     'max_sockets' must be less than FD_SETSIZE.

  INPUTS
     max_sockets   - maximum number of sockets that can be open at once.
                     (4 bytes are allocated for each socket.)
     errno         - pointer to the global int 'errno.'

  RESULT
     retval        - TRUE on success, FALSE on failure.  If 'max_sockets'
                     is greater than FD_SETSIZE, setup_sockets() will
                     fail.  FD_SETSIZE is currently 128 (see
                     <sys/types.h>)

  NOTES
     If you are using a language other than C, you must pass in a pointer
     to a variable that will hold the error numbers.

     In SAS C, errno is a global in lc.lib.  If you don't link with lc.lib
     you will have to declare errno locally.

  BUGS

  SEE ALSO
     cleanup_sockets()
```

```
socket.library/shutdown                              socket.library/shutdown

   NAME
       shutdown -- Shut down part of a full-duplex connection.

   SYNOPSIS
       return = shutdown( s, how )
       D0                 D0 D1

       int shutdown(int, int);

   FUNCTION
       Sockets are normally terminated by using just s_close().  However,
       s_close() will attempt to deliver any data that is still pending.
       Further, shutdown() provides more control over how a connection is
       terminated.  You should eventually use s_close() on all sockets you
       own, regardless of what shutdown() is done on those sockets.

   INPUTS
       s               - socket descriptor.
       how             - 'how' can be one of the following:
                         0    disallow further receives
                         1    disallow further sends
                         2    disallow further sends and receives

   RESULT
       return          - 0 if successful else -1 (errno will contain the
                         specific error).

   EXAMPLE

   NOTES

   BUGS

   SEE ALSO
       s_close()
```

```
socket.library/socket                                  socket.library/socket

   NAME
       socket -- Create an endpoint for communication.

   SYNOPSIS
       s = socket( family, type, protocol )
       D0          D0      D1    D2

       int socket( int, int, int );

   FUNCTION
       socket() returns a socket descriptor for a socket with .

   INPUTS
       family   - This specifies an address format with which
                  addresses specified in later operations using
                  socket should be interpreted.
       type     - Specifies the semantice of communication.
       protocol - Specifies a particular protocol to be used with the
                  socket.

   RESULT
       s        - Returns a (-1) upon failure  ; a socket descriptor
                  upon success.

   NOTES
       Unlike the linkable socket library, this function assumes that
       you have already made a succesful call to 'setup_sockets()'.

   SEE ALSO
```

```
socket.library/strerror                                socket.library/strerror

    NAME
        strerror - Returns a pointer to an error message.

    SYNOPSIS
        #include <string.h>

        error = strerror( error_number )
        D0                D1

        char *strerror( int );

    FUNCTION
        The strerror() function maps the error number to a language-
        dependent Unix error message.

    INPUTS
        error_number - usually the value of errno.

    RESULT
        error        - pointer to the error message.

    NOTES
        This function should eventually be localized.

    SEE ALSO
        <errno.h>,  perror()
```

```
socket.library/syslog                                    socket.library/syslog

   NAME
       syslog - log system messages

   SYNOPSIS
       error = syslog( priority, message )
       D0              A0        D0

       int syslog( int, char * ) ;

   FUNCTION
       syslog() writes the 'message' argument to a console window and/or
       a specified file. The priority field is used to determine which,
       if any, of the above options is used.

       The file "inet:s/inet.config" contains the optional fields:

         filepri         - msgs w/pri >= filepri are sent to file.
         windowpri       - msgs w/pri >= windowpri are sent to window.
         syslogfilename  - the path/name of the file for filepri messages.

       Example:  (Note: this is the exact format to use in the
                        'inet:s/inet.config' file.)

         filepri=5
         windowpri=3
         syslogfilename="t:foobar"

         These entries tell syslog how to deal with the messages it is
         sent.  Note that the smaller the priority number, the greater
         it's priority. Therefore, a message of priority 3 is of greater
         importance than a message of priority 4. If you do not add these
         fields to you 'inet:s/inet.config' file, the default values will
         be used. The default values are:

              windowpri  = 4
              filepri    = 3
              syslogfilename = "ram:syslog.dat"

           The above values mean:

              1. A message sent with a priority >= 4 (4,3,2,1,0) will
                 be sent to BOTH the console window and the file
                 'ram:syslog.dat'.

              2. A message sent with a priority >= 3 (3,2,1,0) will
                 be sent to ONLY the console window.

              3. A message sent with a priority < 4 (5,6,7,8) will
                 be ignored.

   INPUTS
       pri         - an integer value between 0-7
       message     - a string to be written to the console window and/or
                     the syslog file.

   RESULT
       error       - 0 (zero) is returned if everything went well.
                     -1 is returned if anything went wrong.

         If an error has occurred, you need to examine your errno value
         to determine just what went wrong. It is possible, for example,
         to have a write to the console window succeed while a write of
         the same message to the file failed.  The various error values
         that occur during the syslog operation are OR'd together and
         returned in 'errno'. See the file "sys/syslog.h" for the values.

         Example:

         #include<sys/syslog.h>
             ...
          extern int errno ;
          int error ;

          error = syslog( 5, "This is a test\n") ;
          if( error == -1 )
```

```
        {
          if( errno & SYSLOGF_WINDOWOPEN )
          {
             printf("Could not open syslog window\n") ;
          }
          if( errno & SYSLOGF_FILEWRITE )
          {
             printf("Could not write to syslog file\n") ;
          }
        }

NOTES
    1. syslog() will clear the global errno value with each call.

    2. Unlike the Unix syslog() function, the Amiga version does not
       handle printf-style arguments. This will need to be handled in
       the application.

    3. The maximum length of a syslogfilename line in inet.config is
       127 characters. This includes the 'syslogfilename=' portion
       of the line.

       maxlen(path/filename) = 127 - len( "syslogfilename=" )

BUGS
    none known

SEE ALSO
    inet.config (in the AS225 manual)
```

```
socket.library/umask                                socket.library/umask

   NAME
       umask  -- get and set user file creation mask

   SYNOPSIS
       #include <sys/types.h>

       umask = umask( cmask )
       D0             D0

       mode_t umask ( mode_t );

   FUNCTION
       The umask() function sets the file creation mask to 'cmask'
       and returns the old value of the mask.

   RESULT
       -1 will be returned and an error message will be displayed
       if there is a problem reading the current configuration.

   EXAMPLE

   NOTES
       Amiga filesystems, of course, will not use this file creation
       mask.  We use it for NFS, and provide it in case you can think
       of something to use it for.

       The new mask value is not saved to the configuration file. It
       will be reset when the machine is rebooted.

       Currently, the configuration information is stored in memory
       in a configuration structure.  If this structure cannot be
       found, it will be initialized by reading in inet:s/inet.config.
       This may change in the future.


   BUGS

   SEE ALSO
       getumask()
```

8

Writing Localized Applications

by Martin Taillefer

One of the important goals while developing *locale.library* was to keep it simple in order to encourage its use. The result is that applications can be localized with little effort on the programmer's part. Of course, there are some issues to consider in order to do a better localization job.

The central part of localizing an application is getting all of the user-interface running in different languages. *locale.library* provides text catalog management routines to solve this problem. To make use of these catalogs, an application must group all of its strings in one location within the source code, and only refer to them using symbolic names from within the rest of the program. See the sample program at the end of this document for an example of how this can be done easily.

Translated strings vary in length from their originals. This can cause headaches when trying to perform screen layout, or otherwise formatting tables, etc... On the average, it is safe to assume any given English string will double in length when translated to a variety of languages. This fact must always be considered when planning the visuals of a program.

Remember to take into account the time it takes to translate the strings of your application when you plan development schedules. Be flexible, and prepare to maintain a dialog with the translators. It may be necessary to extend the size of some of your gadgets to accommodate special cases. Take this into account from the start, and design your application so these kinds of modifications remain easy.

Localization also involves number output. From country to country, numbers are represented in different ways. As such, it is not correct to assume anything about the length and format of numeric values.

Date format is another localized item. Whenever displaying a date or requesting a date from the user, the FormatDate() and ParseDate() routines from *locale.library* should be used. These routines handle all the details of the variations in date format. Again, it is not safe to assume the length or format of a date string.

A localized application will potentially be viewed by audiences with radically different cultural backgrounds. This is an important point to note when designing icons for an application. Icons designed in the US for example, may seem perfectly obvious to an American, but may be totally meaningless to a Swedish user. Be flexible with your icon designs. Make sure to show them to users of your target countries before releasing your applications.

Something else which is important is naming. When naming programs or features of programs, you should verify that the chosen name is suitable for all of your target markets. Some words may be have totally different meanings, or even be insulting, when viewed in different languages.

Screen Layout

Screen layout is a problem when localizing applications. Strings vary in length from language to language and it becomes difficult to create effective displays that will work well when translated.

When creating a user-interface display on the Amiga, there are two main areas of concern: menus and gadgets.

Menus are easily localized thanks to *gadtools.library*. Its dynamic menu layout code makes adapting to different strings virtually automatic.

Gadgets are not so easily handled. As the text they contain can change, so can their size. And when their size change, it can affect their positioning, which can then start to affect the positioning of surrounding gadgets, which ultimately affect the dimensions of the window.

There are two main ways to handle localization of gadgets:

o make all gadgets large enough to hold any string
o dynamic screen layout

The first approach is simplest and involves making all gadgets large enough to accommodate the largest translated string for that gadget. For example, a gadget containing the string ''Use'' in an English program should not be made just large enough to hold three characters, but instead should be large enough to contain the longest translation of the word ''Use''. The 2.1 system preferences editors make use of this approach to localize their gadgetry.

The second approach to handle gadgets is by doing font sensitive layout. This involves determining exactly the size of all strings used to create a display, then to scale and position every component so it can deal with the largest strings out there. Font sensitive layout is normally thought of as a way of accomodating different fonts, but it also serves to accommodate different strings. The system 2.1 *Calculator* is an example of a font sensitive application.

Adapting Gadtools

GadTools' NewMenu and NewGadget structures both have Label fields requiring a pointer to a string. The string is used to build the user-interface component. Many programs currently declare a static array of these structures to pass them to either CreateMenus() or CreateGadget(). Static initialization will not work however within a localized application due to the fact the label strings are not known at compile-time, and are instead obtained at run-time from disk-based catalogs.

Localizing GadTools Gadgets

Here is a routine that lets you easily localize GadTools gadgets. Instead of calling the GadTools CreateGadget() routine, simply call CreateLocGadget() instead.

```
extern struct Catalog *catalog;

struct Gadget *CreateLocGadget(ULONG kind, struct Gadget *previous,
                               struct NewGadget *newGad, ULONG tag, ...)
{
struct NewGadget ng;

    ng = *newGad;
    ng.ng_Label = GetCatalogStr(catalog,
                                (LONG)ng.ng_Label,
                                GetAppStr((LONG)ng.ng_Label));

    return (CreateGadgetA(kind,previous,&ng,&tag));
}
```

This function assumes you have an opened message catalog called ''catalog'', and that your application has a function called GetAppStr() that returns the built-in string of the application given a string ID number.

Now the trick with this function is that when you declare your NewGadget structure, you initialize the ng_Label field to contain the string ID of the desired string, instead of a pointer to the string itself. CreateLocGadget() will then replace the string ID by an actual string pointer and will call the real GadTools routine to create the gadget.

Localizing GadTools Menus

Localizing menus is a bit trickier than gadgets because the CreateMenus() call accepts an array of NewMenu structures as a parameter instead of a single structure at a time like CreateGadget(). Still, the solution remains relatively simple:

```
struct Menu *CreateLocMenus(struct NewMenu *newMenus, ULONG tag, ...)
{
UWORD           i;
struct NewMenu *nm;
struct Menu    *menus;

    i = 0;
    while (nm[i++].nm_Type != NM_END) {}

    if (!(nm = AllocVec(sizeof(struct NewMenu)*i,MEMF_CLEAR|MEMF_PUBLIC)))
        return(NULL);

    while (i--)
    {
        nm[i] = newMenus[i];

        if (nm[i].nm_Label != NM_BARLABEL)
        {
            nm[i].nm_CommKey = GetCatalogStr(catalog,
                                             (LONG)nm[i].nm_Label,
                                             GetAppStr((LONG)nm[i].nm_Label));

            nm[i].nm_Label = nm[i].nm_CommKey+2;

            if (nm[i].nm_CommKey[0] == ' ')
                nm[i].nm_CommKey = NULL;
        }
    }
    if (menus = CreateMenusA(nm,&tag))
    {
        if (!(LayoutMenus(menus,visualInfo,GTMN_NewLookMenus,TRUE,
                                           TAG_DONE)))
        {
            FreeMenus(menus);
     menus = NULL;
 }
    }

    FreeVec(nm);

    return(menus);
}
```

Like the gadget example above, this function expects to have both an open catalog and a GetAppStr() function available. In addition, this function expects to find the menu strings in a specific format. That is, all strings must be preceded by their keyboard shortcuts and a NULL byte. For example:

''X\0Cut''
''C\0Copy''
''V\0Paste''

This is compact, and lets the keyboard shortcuts be easily localized and associated with the menu item. If a menu item has no keyboard shortcut, it needs to be specified using a leading space such as:

'' \0Erase''

As with gadgets, the trick with this function is to initialize all the nm_Label fields to be string IDs instead of direct string pointers. CreateLocMenus() then allocates a new array, replacing the string IDs with string pointers, and passes the result to GadTools for the actual menu strip creation and layout.

Character Sets

A character set defines the meaning and graphical representation associated with a sequence of numbers. From the beginning, the Amiga has used the ECMA-Latin 1 character set which is a superset of regular ASCII designed by the European Computer Manufacturer's Association. This character set is sufficient to represent alphabets used by languages spoken in Western Europe.

Commodore has never endorsed character sets other than ECMA-Latin 1, which means that alphabets used in Eastern Europe, in the Middle-East, or in Asia, cannot be represented in a standard way on an Amiga. Designing fonts with the needed characters in them is not sufficient, because without a formal standard, there is no way to guarantee that a given character will be used in every variation of a font for a given alphabet. To clarify, without a standard, it is not possible to assume that the Greek Omega letter will be represented by the same character value from one Greek font to the next.

locale.library assumes nothing about the character set being used. The character set is dependant on the current language driver and catalog. At this time, the only character set formally recognized is ECMA-Latin 1. New character sets can be specified easily.

It is possible that new character sets may require more than one byte of storage per character. This would be the case for the Unicode character set. Applications written now only need to deal with the current 1-byte character set. In the future, applications may want to support new extended character sets as they are standardized upon by Commodore.

Narrator Device and Translator Library

The current *narrator.device* and *translator.library* are geared towards English speech. Phonemes are missing in *narrator.device* in order to allow true multi-lingual speech, while *translator.library* only knows how to translate English text into phonemes.

The first localization release will not include updated versions of these system modules. Future releases of the Amiga operating system may provide these.

Catcomp: The Catalog Compiler

CatComp is a program to handle *locale.library*'s message catalog creation and maintenance. Message catalogs are IFF files read by *locale.library*. They contain all the text strings used by an application. By providing several of these catalog files, an application can use *locale.library* and transparently adapt itself to the user's preferred language.

CatComp reads and processes two kinds of input files, and produces four types of output files.

The first kind of file read by *CatComp* is a catalog description file. That file describes all the strings used by an application. There is one such file per localized application. In this file, you assign numbers to all the application strings as well as specify their minimum and maximum lengths.

The second kind of file read by *CatComp* is a catalog translation file. That file contains all strings from the application translated to one language. There is one catalog translation file per language supported by a localized application.

CatComp can produce four different kinds of output files. The first is an IFF catalog. These are regular IFF files. There is one catalog produced for every catalog text file successfully processed. Catalogs are read directly by *locale.library* whenever the OpenCatalog() call is made.

The second type of file output by *CatComp* is an empty translation file. This is useful when starting a translation. It lets *CatComp* convert a catalog description file into a ready to fill-in translation file.

The third and fourth type of files output by *CatComp* are C and Assembly language programming source files. These are used by application writers to convert catalog description files into source files that can be directly used in their applications.

Invoking CatComp

CatComp can only by run from the Shell under Kickstart V37 or beyond. Its template is:

DESCRIPTOR/A,TRANSLATION,CATALOG/K,CTFILE/K,CFILE/K,ASMFILE/K,VB=VERBOSITY/N

DESCRIPTOR/A
Specifies the name of a catalog description file. Typically, the file extension for catalog description files is *.cd*

TRANSLATION
Specifies the name of a catalog translation file. Typically, the file extension for catalog translation files is *.ct*

CATALOG/K
Specifies the name of the IFF catalog file to produce. When this option is specified, a translation file must have been given in the previous argument.

CTFILE/K
Specifies the name of the blank catalog translation file to produce. This option only requires a catalog descriptor file be provided, there is no need for a translation file.

CFILE/K
Specifies the name of the C-language source file header to produce. This option only requires a catalog descriptor file be provided, there is no need for a translation file.

ASMFILE/K
Specifies the name of the Assembly-language source file header to produce. This option only requires a catalog descriptor file be provided, there is no need for a translation file.

VERBOSITY/N
Specifies the amount of information *CatComp* should output while doing some processing. The lower this number, the less *CatComp* will output messages. Not providing this option causes *CatComp* to output every message it could.

Catalog Description Files

Catalog description files completely describe the strings used by an application. The format for these files is basically a series of two line entries separated by an arbitrary number of comment lines. The EBNF (Extended Backus Naur Formalism) specification for catalog description files is:

```
{# command}
{; comment line}
<string name> "(" [string id] "/" [min string len] "/" [max string len] ")"
[string]
```

The first line indicates special commands that provide control over the generation of C and Assembly source files. See the section on C and Assembly source files below for further information.

<string name> is the symbolic name of the string. Following the name comes three optional numbers enclosed in parenthesis and separated with slashes. The first number specifies the string's ID value. This is the ID value used to request this string when using the GetCatalogStr() call in *locale.library*. The second number specifies the minimum length for the string, while the third number specifies the maximum string length.

The next line of an entry contains the actual string. Within the catalog description file, the strings are the same as the default strings built into the application. The strings can use standard C-language backslash ('\') escape sequences.

Finally, any string entry can be separated from other entries by comment lines. Comment lines start with a semicolon. There can be any number of comment lines between two entries. It is not allowed to put comment lines between the two lines of a same entry.

Here's an example of two strings in a catalog description file:

```
MSG_HELLO (0/4/50)
This is a test string for the world to see
;
MSG_BYE (1/14/47)
This is another test string
```

The first string is called MSG_HELLO, has 0 as ID value, can be as short as 4 characters and as long as 50. The string actually associated with the name follows on the next line. Then comes a comment line. Following that is the definition of the second string called BYE_WORLD_STR that has 1 as ID value, can be as short as 14 characters and as long as 47.

As mentioned above, all three numbers in an entry are optional. If the string id number is not specified, then the same id as the previous string in the file plus one is used. If the first string defined has no id value, the starting id value is 0. If the minimum string length is not specified, it is assumed to be 0. And finally, if the maximum string length is not specified, it is assumed to be unlimited.

A note on style here. It is relatively important to choose descriptive symbolic names for the strings. These names will be viewed by the translators and should be as meaningful as possible. The following conventions are suggested:

- List all names in capital letters. This will make it clear they are constants as this is the convention used in all Amiga include files.
- Prefix each name with the string MSG_. This will make it clear to the programmer that a given constant is in fact a string ID value.
- Append the string _GAD to strings that are used for gadget labels.
- Append the string _MENU to strings that are used for menu titles. For example, MSG_EDIT_MENU.
- Expand the path leading to a menu item when specifying strings that are used for menu items. For example, for the Cut item in the Edit menu would be written as MSG_EDIT_CUT.

Catalog Translation Files

Catalog translation files contain all the strings of an application translated to a different language than the default. The files look very similar to catalog description files, except they do not include the string id, minimum string length and maximum string length specifications. The EBNF specification for catalog translation files is:

```
{# command}
{; comment line}
<string name>
[string]
```

The first line indicates special commands that describe attributes of the catalog file. There are currently three supported commands. ''version'' lets you specify a 2.0-style version string for the catalog. ''language'' lets you list the language that this catalog is in. This string should itself be in the given language and not in English.

For example:

```
MSG_HELLO
Ceci est une chaine test pour être vue
;
MSG_BYE
Ceci est une autre chaine test
```

On the first line you list the symbolic name of the string, and on the second line the translated string. The symbolic name is the same as the related entry in the catalog description file. *CatComp* uses this name to associate entries from translation files with entries in description files. It can then validate the strings in the translation files by ensuring they are of the correct length, etc.

Escape Sequences Supported

Regular C-language escape sequences can be specified in strings, along with a few additions:

\a	inserts an audible bell character (ASCII 7)
\b	inserts a backspace character (ASCII 8)
\c	inserts a control sequence introducer (ASCII 155)
\e	inserts an escape character (ASCII 27)
\f	inserts a formfeed character (ASCII 12)
\n	inserts a newline character (ASCII 10)
\r	inserts a carriage return character (ASCII 13)
\t	inserts a tab character (ASCII 9)
\v	inserts a vertical tab character (ASCII 11)
\xNN	inserts NN, where NN specifies an ASCII code in hexadecimal
\NNN	inserts NNN, where NNN specifies an ASCII code in octal

Formatted Output Commands

CatComp parses strings for C-language formatting commands as used in the printf() function. It ensures that the number and type of such commands are the same in both the description file and the translation files. This guarantees that the application stack frame will not be misinterpreted due to incorrect translations of formatting commands.

CatComp warns you if you attempt to use any non-C formatting commands. The commands that *CatComp* does understand are:

```
%b %c %d %e %E %f %g %G %i %o %p %s %u %x %X
```

CatComp also knows about the ordering formatting command supported by RawDoFmt() whenever *locale.library* runs in the system, or by locale's FormatString() routine. The ordering command lets you specify formatting commands within a formatting string in a different order than in the original string, while still accessing the stack frame correctly. *CatComp* validates the ordering information and ensures argument types match. See the documentation *locale.library*/FormatString().

Specifying the argument position lets the order of the % commands within your strings without affecting how the program performs. Given a string in a catalog description file such as:

```
MSG_AVAIL_MEM (//)
FAST: %1U, GRAPHIC: %1U
```

This string could be translated in French as:

```
MSG_AVAIL_MEM (//)
GRAPHIQUE: %2$1U, AUTRE %1$1U
```

Using the first string, the output of the program might look like:

```
FAST: 1234, GRAPHIC: 5678
```

while the translation would output:

```
GRAPHIQUE: 5678, AUTRE: 1234
```

Validation

CatComp enforces the syntax of catalog description files and catalog translation files very strongly. It also ensures that the same number of C-language % command appear in both the description and the translation file. This guarantees the integrity of the application stack-frame when using translated string in printf() statements.

Most errors detected by *CatComp* are fatal and cause the program to abort. Errors are reported to the console with a descriptive error message, a filename, and if needed a line and column number. Non-fatal errors (warnings) and also sent to the console, but they do not cause the program to abort. The printing of these warning messages can be suppressed using the VERBOSITY command-line option.

Possible Errors

Here is a list of the errors and warnings that *CatComp* can produce, along with an explanation of what went wrong. Most errors indicate the file, line and column where the error occurred to help in solving the problem.

ERROR: string line for token <name> not found
A given token was not followed by a string

ERROR: token not found
No token was found on a line in a catalog description file. Comment lines must start with ";", any other line must have a valid token definition on it.

ERROR: '(' expected
There was no number section after a token in a catalog description file. The number section must start with a (, followed by three optional numbers separated by slashes, and terminated by a).

ERROR: ')' expected
There was no) after a number section in a catalog description file. The number section must start with a (, followed by three optional numbers separated by slashes, and terminated by a).

ERROR: '/' expected
There was no slash found within a number section after a token in a catalog description file. The number section must start with a (, followed by three optional numbers separated by slashes, and terminated by a). So there must always be two slashes specified.

ERROR: garbage characters after token <name>
There was no number section after a token in a catalog description file, and garbage characters were found instead, The number section must start with a (, followed by three optional numbers separated by slashes, and terminated by a).

ERROR: <name> is not a valid token
A token in a catalog description file was composed of invalid characters. A token must start with a letter and can be followed by letters, numbers and underscores.

ERROR: token <name> not found
A token specified in a catalog description file was not present in a translation file

ERROR: string too short for token <name>
A string in a translation file is shorter than the minimum length specified in the description file.

ERROR: string too long for token <name>
A string in a translation file is shorter than the minimum length specified in the description file.

ERROR: negative value for minimum length
The minimum string length specified for a token must be positive.

ERROR: negative value for maximum length
The maximum string length specified for a token must be positive.

ERROR: non-positive value for % ordering
The position information for a % formatting command must be positive and greater than 1.

ERROR: % ordering value too large
The position information for a % formatting command is greater than the number of formatting commands provided

ERROR: % size incorrect
The size specifier for a % formatting command in a translation file does not match the size in the description file.

ERROR: % command does not match
The type specifier for a % formatting command in a translation file does not match the size in the description file.

ERROR: token <name> defined multiple times
A token was defined multiple times in either a description or a translation file.

ERROR: id <number> already used for token <name>
An attempt was made to reuse an ID value twice within a description file

ERROR: no command found after '#'
Command lines start with # and are followed by a command.

ERROR: <name> is not a valid command after '#'
A command specified after # is invalid.

ERROR: <number> is not a valid codeset value
An incorrect codeset value was specified for a #codeset command.

ERROR: couldn't write catalog <name>
An error occurred while writing the catalog file

WARNING: <name> is an unknown formatting command
An unknown % formatting command was specified. *CatComp* knows only of C-language formatting commands, anything else will be flagged with this warning.

WARNING: string for token <name> matches string in descriptor A string within a translation file matches exactly the original string from the description file. This may mean that the string was not translated.

C and Assembly Source Files

CatComp has the ability to output C and Assembly language source file headers. The intent of this is to let application programmers manipulate a single catalog description file and have source files generated for them automatically so they can include the strings in their programs.

To generate these files, you need to give *CatComp* a descriptor file, and either a C or an ASM output file name using the CFILE/K and ASMFILE/K command-line options. The resulting files will be standard C and/or ASM sources file headers that be used easily in application code.

Sample Use

This section presents sample uses of *CatComp* with example command-lines.

Assume you have a catalog description file for an application called *app.cd.*

To test if this file is a valid *.cd* file, type:

```
CatComp app.cd
```

and *CatComp* will respond with either some error messages, or with a message saying that *app.cd* is a valid descriptor file.

To do a translation of a *.cd* file, you need a *.ct* file. Such a file can be generated by doing:

```
CatComp app.cd CTFILE app.ct
```

This will create a blank translation file called *app.ct.* You can then load *app.ct* in a standard text editor and proceed to translate the strings it contains.

Once a *.ct* file is done being translated, it must be converted in an IFF catalog file. This is done by doing:

```
CatComp app.cd app.ct CATALOG app.catalog
```

This will create a file called *app.catalog*.

Example Application

Here is a complete localized application using *CatComp*. All needed files are shown.

sample.cd

This is the catalog description file which lists all the strings used by the application.

```
MSG_HELLO (//)
Hello World!\n
;
MSG_BYE (//)
Goodbye World!\n
```

sample.ct

Here is a sample French translation file for the application.

```
## version $VER: sample.catalog 38.1 (29.7.91)
## codeset 0
## language francais
;
MSG_HELLO
Bonjour Le Monde!\n
;
MSG_BYE
Adieu Terre Ingrate!\n
```

samplestr.h

This is the file that *CatComp* outputs when processing the *samplestr.cd* file.

```
#ifndef TEXTTABLE_H
#define TEXTTABLE_H

/****************************************************************************/


/* This file was created automatically by CatComp.
 * Do NOT edit by hand!
 */

#include <exec/types.h>

/****************************************************************************/


#define MSG_HELLO 0
#define MSG_HELLO_STR "Hello World!\n"

#define MSG_BYE 1
#define MSG_BYE_STR "Goodbye World!\n"

/****************************************************************************/
```

```
#ifdef STRINGARRAY

struct AppString
{
    LONG   as_ID;
    STRPTR as_Str;
};

struct AppString AppStrings[] =
{
    {MSG_HELLO,MSG_HELLO_STR},
    {MSG_BYE,MSG_BYE_STR},
};

#endif /* STRINGARRAY */

/****************************************************************************/

#endif /* TEXTTABLE_H */
```

texttable.h

This is the header file for a general-purpose catalog handling routine. Localized programs will generally contain something similar to this. The include file ''samplestr.h'' gets generated by *CatComp* using the sample *.cd* file above.

```
#ifndef SAMPLE_TEXTTABLE_H
#define SAMPLE_TEXTTABLE_H

/****************************************************************************/

#include <exec/types.h>
#include "samplestr.h"

/****************************************************************************/

struct LocaleInfo
{
    APTR li_LocaleBase;
    APTR li_Catalog;
};

/****************************************************************************/

STRPTR GetString(struct LocaleInfo *li, LONG id);

/****************************************************************************/

#endif /* SAMPLE_TEXTTABLE_H */
```

texttable.c

This is the source file for a general-purpose catalog handling routine. Localized programs will generally contain something similar to this.

```
#define STRINGARRAY

#include <exec/types.h>
#include <libraries/locale.h>
#include <clib/locale_protos.h>
#include <pragmas/locale_pragmas.h>
#include "texttable.h"

#define LocaleBase li->li_LocaleBase
#define catalog    (struct Catalog *)li->li_Catalog

STRPTR GetString(struct LocaleInfo *li,LONG id)
{
UWORD  i;
STRPTR local = NULL;

    i = 0;
    while (!local)
    {
        if (AppStrings[i].as_ID == id)
            local = AppStrings[i].as_Str;
        i++;
    }

    if (LocaleBase)
        return(GetCatalogStr(catalog,id,local));

    return(local);
}
```

sample.c

```
/* sample.c */
#include <exec/types.h>
#include <exec/libraries.h>
#include <libraries/locale.h>

#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/locale_protos.h>

#include <pragmas/exec_pragmas.h>
#include <pragmas/dos_pragmas.h>
#include <pragmas/locale_pragmas.h>

#include "texttable.h"

/*****************************************************************************/


extern struct Library    *DOSBase;
extern struct Library    *SysBase;
       struct LocaleInfo li;

/*****************************************************************************/


#define LocaleBase li.li_LocaleBase
#define catalog    li.li_Catalog

/*****************************************************************************/


VOID main(VOID)
{
    if (LocaleBase = OpenLibrary("locale.library",38))
        catalog = OpenCatalogA(NULL,"sample.catalog",NULL);

    PutStr(GetString(&li,MSG_HELLO));
    PutStr(GetString(&li,MSG_BYE));

    if (LocaleBase)
    {
        CloseCatalog(catalog);
        CloseLibrary(LocaleBase);
    }
}
```

The A3000 Local Bus Expansion Slot Specification

A High Performance Expansion Slot specific
to Amiga 3000 family computers

Document Revision 2.0

1991 DevCon Release

by
Scott Schaeffer
and
Dave Haynie

July 16, 1991

Copyright © 1991 Commodore-Amiga, Inc.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 Intended Audience 1-1
1.2 Why a Local Bus Slot? 1-1
1.3 Why an Expanson Bus Slot? 1-2
1.4 The Amiga 3000 Family 1-2

CHAPTER 2 FUNCTIONALIY AND DESIGN GUIDELINES

2.1 Slave Devices 2-1
2.2 Master Devices 2-2
2.2.1 Primary Bus Mastership 2-2
2.2.2 Secondary Bus Mastership 2-3
2.3 Clock Generation 2-4
2.4 Local Bus Design Criteria 2-5

CHAPTER 3 SIGNAL DESCRIPTIONS

3.1 Power Connections 3-1
3.2 System Initialization 3-1
3.3 68030 Signals 3-2
3.4 Bus Arbitration Signals 3-4
3.5 Other Local Bus Signals 3-4
3.6 Clocks 3-5
3.7 New Amiga 3000T Signals 3-5
3.8 New Amiga 3000+ Signals 3-6

APPENDICES

A.1 Local Bus Connector Pinout.......... A-1
A.2 CPU/Clock Jumpers.......... A-5
A.2.1 A3000 Jumpers.......... A-5
A.2.2 A3000T Jumpers.......... A-5
A.2.3 A3000+ Jumpers.......... A-6
A.3 References.......... A-8

CHAPTER 1

INTRODUCTION

"What works for me might work for you"
-Jimmy Buffett

This document describes the Amiga 3000 Local Bus Expansion Slot, also known as the Amiga 3000 Coprocessor Slot. This expansion slot is designed to provide high speed access to the Amiga 3000's local, or 68030, bus. This slot is intended for high speed expansion devices that are generally very specific to the 68030 bus or need direct access to the local bus for other reasons. Such devices include alternate 680x0 family processors, cache memory boards, high speed bursting ram expansion, and similar things.

1.1 Intended Audience

This document is written for hardware engineers interested in designing cards for the Amiga 3000's Local Bus Slot. A good level of microcomputer systems design knowledge is necessary to get much meaning out of these pages. Especially important is familiarity with the 68030 processor hardware conventions, as most of the Local Bus Slot is based directly on the 68030 processor bus.

1.2 Why a Local Bus Slot?

The local bus slot was originally introduced on the Amiga 2000, and has served its intended purpose quite well on that system. We expect the A3000's Local Bus Slot to provide similar functionality on the A3000.

The main use for the Local Bus Slot is the addition of a single, very tightly coupled expansion device, which is typically a high speed CPU, cache, or memory board. An alternate 680x0 family device, such as a 68040 processor, needs access to every 68030 signal in order to properly replace the 68030 processor as a host for AmigaOS or UNIX. Cache memories or extremely fast "Fast" memory needs direct access to the 68030 bus to run as tighly coupled as possible to the 68030 processor on the A3000's motherboard. Most other devices expansion devices are expected to be designed as Zorro II or Zorro III expansion cards.

In any system design, you can make good arguments for a general purpose expansion bus, and good arguments for an extendable local bus. The Amiga philosophy is that both of these approaches are correct, and seve complementary needs. Any of the stated candidates for the Local Bus Slot are by their very nature tightly coupled to the A3000's system bus, which is of course based on the 68030 bus. They are not expected to work in future Amiga systems, which could easily have completely different local buses, and therefore, different local bus slots. Also, it's impractical to provide a large number of such slots, since the local bus itself has tight electrical limits on expandability, yet any buffers imposed between the local bus and Local Bus Slot devices can reduce the high performance we're striving for. So a single local bus slot is provided.

1.3 Why an Expansion Bus Slot?

Most Amiga 3000 add-on cards belong in an Expansion Bus Slot. First of all, since there is only one Local Bus Slot, it stands to reason that only one such device can be added to any system, while at least four Expansion Bus (Zorro III) slots are available on any A3000 family computer, and can be expected on any more advanced architure Amiga in the future. The Zorro III bus doesn't permit the same degree of tight coupling that the Local Bus Slot does, so its not capable of supporting cache or other zero wait-state memory, and it can't support a direct-replacement 68040. It does permit reasonable speeds, interrupts, bus locking, etc. so it is the place for high performance I/O devices, moderate speed add-on memory boards, processor devices such as DSP, Video, or RISC devices that coexist with the main processor, etc. And of course, devices that are happy as slower 16 bit peripherals can be implemented as Zorro II cards and have the advantage of working in all Amiga computers (including the A500 and A1000 with the proper 3rd party bus adaptor or backplane). Details on the Zorro III bus are available in *The Zorro III Expansion Bus Specification*, while the Zorro II bus is described in *The A500/A2000 Technical Reference Manual.*

1.4 The Amiga 3000 Family

There are three basic Amiga 3000 class machines: the original Amiga 3000, the Amiga 3000T, and the Amiga 3000+. Some enhancements to the original A3000 Local Bus Slot were added to the A3000T, mainly signals associated with the jumper-less clock takeover mechanism. These are described in section 3.7. Additional enhancements were added for the A3000+; these are described in section 3.8.

CHAPTER 2

FUNCTIONALITY AND DESIGN GUIDELINES

"Time and distance are out of place here"

-REM

The Amiga 3000 Local Bus Expansion Slot provides signals to implement both slave and master devices. Memory devices, including cache, are bus slaves, while CPU devices such are 680x0 accelerator boards are bus masters. Any card may, at times, be either slave or master, and in a few cases, both at once.

2.1 Slave Devices

A slave device is a device that responds to the current local bus master. The local connector provides direct access to all local bus signals, which include all 68030 signals, plus a few additional Amiga 3000 specific lines to allow a Local Bus Slot device to control the A3000's Local Bus. Local Bus Slot slaves are not autoconfigured as are Zorro III bus devices, but instead are fixed in the address range from $08000000 to $0fffffff. The A3000 local bus controller, the Fat Gary chip, provides a decode of this space on the signal /RAMSLOT which is valid at address time. This signal may be ignored and the address decoded by logic on the board if speed is an issue. Local slave devices should also support the signal /CIIN if they contain uncachable data. The Amiga OS expects anything mapped starting at $08000000 to be memory, and in fact, the fastest memory in the A3000 system. The OS will automatically size and link in any memory it finds here, and place it in the system as the highest priority memory available. To support control registers or other memory mapped resources on a Local Bus Card, locate them at least 512K above the $08000000 base, and the OS will ignore them.

A signal named /WAIT is provided for cache support. Asserting this signal will disable address decoding of onboard Fast RAM by the RAMSEY chip and Zorro II/III bus accesses by the BUSTER chip. Constraints imposed by the 68030 allow only 18ns to determine a cache hit. It is often more feasible to assert /STERM before knowing whether the cycle is a cache hit and rerunning the cycle via /HALT and /BERR if it is a miss. To achieve this functionality any decoding of the first cycle by RAMSEY or BUSTER must be disabled by asserting /WAIT less than 10ns after address valid. If the cycle is determined to miss a rerun is initiated and wait deasserted for the secondary cycle. Assertion of /WAIT will keep /STERM, /CBACK, etc. tristated by BUSTER or RAMSEY and may be controlled by the cache control logic.

2.2 Master Devices

A Master Device is, of course, a device which masters the local bus, replacing the functions of the 68030 during its period of mastership. Bus mastership may be accomplished two ways depending on the desired functionality. The first mode, called *primary mastership,* totally disables the motherboard 68030 and its arbitration logic, essentially replacing the 68030 with the local slot device. Such a device takes over full arbitration responsibility for the whole system, and must service all interrupts. The second mode, called *secondary mastership,* allows the on-board 68030 and the local bus accelerator board to share the bus, permitting multiprocessor capabilities or more traditional DMA from the local bus board. This protocol permits very fast switching between the local slot master and the motherboard 68030. In this mode, the 68030 is still responsible for bus arbitration.

2.2.1 Primary Bus Mastership

The primary bus mastership, or arbitration takeover mode, requires less logic to implement and may be preferred in most implementations. In the absence of multiprocessing software support, this is the mode of choice, and corresponds to the way in which most A2000 local slot cards worked.

The local bus card asserts /CBR at power on to the motherboard which in turn asserts /BR to the motherboard 68030. Upon receiving /BG30 from the motherboard the local card asserts /BOSS. Logic on the motherboard uses /BOSS to force /BGACK30 low to the 68030 only and not the shared local bus /BGACK. In addition the assertion of /BOSS tristates /BG on the motherboard and in turn the local card should untristate and source its /BG. The local card is now the default bus master and arbiter -- it must provide arbitration for the local bus, based on the 68030 bus arbitration rules. The onboard 68030 is bus arbitrated away and never regains the bus. Pal equations to implement this are given in *Figure 2-1*, all pal equations are active high and should be inverted in the output stage of the pal for active low assertion.

Actual timing for takeover mode is not given since all signals are inherently asychronous. The untristating of /BG should be later than the tristating of /BG on the motherboard to minimize contention on that signal.

```
/* Always assert Coprocessor Bus Request. */
CBR         =     'b'1;

/* The Boss Signal. "poweron_reset" is a signal sourced by the local card and
   is asserted for a few hundred nanoseconds after poweron. This clears the
   feedback path on the pal and may be generated by an RC network which is
   slewrate cleaned by a schmitt trigger device. */
BOSS        =     BG30 # BOSS & !poweron_reset;

/* The Grant Signal. "local_card_bg" is sourced by the local card in
   compliance to  the operation of arbitration defined in the 68030 users
   manual. */
BG          =     local_card_bg;

BG.oe       =     BOSS;
```

Figure 2-1: Primary Mode Takeover PAL Equations

2.2.2 Secondary Bus Mastership

The secondary bus acquisition mode uses the 68030 arbiter to provide cycle arbitration between the local slot card and the motherboard DMA. Since the 68030 provides only a single bus request input, a scheme referred to as fast arbitration is used between the local card and all other DMA sources, which are controlled via the BUSTER chip. Bus request is an open collector line which is time multiplexed between the local card and BUSTER. On a positive edge of CPUCLK, BUSTER will assert /BR if it requires the bus and /BR is not already asserted by the local bus card. On the negative transition of CPUCLK, the local bus card may assert /BR if it is not already asserted by BUSTER. This scheme allows both masters to share a single bus request and also requires only 20ns to resolve the master arbitration. This mode is very efficient but forces the local card to use high speed logic since the time between clock edges is so short. It is *strongly* suggested that the above logic be incorporated in a 7.5ns or faster registered PAL connected to a F38 open collector device. Clock skew between the clock to this pal and CPUCLK is critical and it is advised that the local card generate and source clocks to the motherboard, especially if CPUCLK is needed elsewhere on the card; load on CPUCLK at the Local Bus Slot must be kept to a minimum to avoid mucking with the local bus timing. Skew between CPUCLK and the clock driving this pal should be less than 2ns. The PAL equations for /BR are given in *Figure 2-2.*

After receiving /BG from the motherboard 68030 the local card drives /BGACK (open collector) and assumes mastership of the bus. It may keep the bus for multiple cycles but should not hog the bus for extended periods unless it relinquishes the bus when DMA request is asserted by BUSTER. Hogging the bus may cause adverse operation of the A3000. Be aware that the local bus signals must be tristated (/AS, /DSACK, Address, Data, etc.) prior to deasserting /BGACK. In addition the local card must not drive the bus or /BGACK until /AS, /DSACK, /BERR, /HALT, etc. have deasserted. In other words, standard 680x0 bus arbitration rules apply here. It is generally assumed that any secondary bus master mode Local Bus Slots device will be attempt to prevent undue bus hogging at the design level. This implies either a coprocessor

```
/* This is the BR_LOCAL signal, which is an active output fed into a 74f38, or
   other open collector equivalent buffer, to produce /BR.  /BR is the raw bus
   request from the local bus connector (keep this trace short on the local
   card).  WANTBUS is the request from the local card which deasserts after it
   sees BR_LOCAL asserted.  BGACK_LOCAL is /BGACK asserted by the local card.
   RESET is any reset signal used to prevent poweron latchup of BR_LOCAL. */

BR_LOCAL.d    =      !BR & !BR_LOCAL & WANTBUS
              #      BR_LOCAL & !BGACK_LOCAL & !RESET;
```

Figure 2-2: Secondary Mode Takeover PAL Equations

device of some kind that makes only periodic requests of the bus, or a CPU subsystem that contains its own local cache or memory. Any device that requires the mastering the local bus for very long periods of time should be a primary mode bus master.

It is extremely important that the bus master timing from the local card emulate operation of a 25MHZ or 16MHZ 68030 chip *exactly* as defined by the 68030 user manual. Future enhancements to the A3000 motherboard chips may incorporate rerun cycles and currently incorporate burst cycles and cache coherency signals (/CIIN). Signals received by the local card from the motherboard provide only the minimum setup and hold required by a 68030. Do not assume for example that data setup from the motherboard will not significantly change , ie. data setup from Fast RAM on subsequent cycles of a burst is much less than a typical non-burst cycle.

In addition to the 68030 specifications, the A3000 local bus adds one additional constraint. Slave devices should should hold data through the end of cycle, regardless of whether the cycle is terminated by /STERM or the /DSACK lines. In other words, only the current bus master really knows when a cycle has completed. Obviously, burst cycles are an exception to this; they follow the standard 68030 rules in dealing with data hold times.

2.3 Clock generation

The A3000 motherboard provides links to disable generation of CPUCLK and CLK90. This allows the local slot card to maintain better clock skew relationships between its own logic and that of the motherboard, and is especially important if the Local Bus card depends upon being synchronous to the A3000 motherboard clocks -- just as with the A2000 local bus slot, both synchronous and asynchronous designs are possible, but synchronous designs are much more timing critical than on the A2000.

If a local bus card drives the clock lines the appropriate jumpers must be moved on the motherboard. CLK90 must be a clock 90° out of phase from CPUCLK. It is typically generated from a 5 tap 25ns delay line where CLK90 is the 10ns tap when running the system at 25Mhz and CLK90 is the 15ns tap when at 16Mhz. Note that these clocks are fed through a 74f08 to provide clocking to the motherboard. If the skew generated by the f08 is unaccepable (as in fast arbitration) the socketed f08 may be removed and replaced by a header which shorts the appropriate inputs to outputs, though it is the responsibility of the Local Bus card at that point to make sure acceptible versions of CPUCLK and CLK90 exist everywhere on the motherboard. Again careful layout of the clock circuitry is essential for reliable operation.

2.4 Local Bus Design Criteria

Any design which plugs into the local bus connector must comply to some basic design rules.

- Due to inductance in the 200pin connector to VCC and GND it is very imporant to provide ample bypass capacitance in order to maintain good dc VCC and GND levels.
- All signals from this connector are unbuffered and should not be heavily loaded. A good rule is 2 TTL loads. In addition receivers and drivers should be located near the connector and any connector signal should not run over 4 inches in length from the connector before entering or leaving a driver or receiver.
- Clock generation is especially critical. Keep traces short, ECL routing rules should be followed if possible . Fan out multiple clocks from a single die to minimize loading per clock . Light damping resistors minimize radiation but cause clock distortion, so tune the values carefully.
- Keep in mind current draw on the A3000 is tight so use CMOS and powerdown DRAM modes when possible. New FCT devices use significantly lower current than F and run faster. A local bus card should draw no more than 2Amps @ 5VDC.
- Be aware of heat dissipation issues especially on very high speed microprocessors.
- Noise test local bus cards and ensure good ac signal quality since a nasty signal will get nastier after passing through an inductive connector to the motherboard. And keep in mind that any noise a local card injects into the system will make the entire A3000 less reliable.
- Local bust card mounting holes are plated through to ground on the A3000 motherboard and provide an additional low inductance path to ground. Use this path to minimize ground bounce relative to the motherboard.

CHAPTER 3

SIGNAL DESCRIPTIONS

"There's a name for it
And names make all the difference in the world"
-David Byrne

The signals on the Local Bus Slot can be broken down into several categories. Some of these are in common with the 68030, some are specific to this slot. The actual pinout of the connector is detailed in the appendix.

3.1 Power Connections

These signals provide digital supply levels to the local bus card. There are quite a few of both levels on the physical connector; generally at least one for every two or three signal pins.

Digital Ground (GND)
This is the digital supply ground used by all digital devices in the system. The local bus card gets GND though its mounting posts as well as the connector pins.

Digital Supply (+5VDC)
This is the digital supply. This is specified as +5VDC ± 5%. The system power budget allocates up to 2Amps for this slot.

3.2 System Initialization

These signals are driven by the A3000 motherboard logic to initialize the system; local bus cards should listen as appropriate.

/RESET

This is an open-collector signal driven by the system reset logic or, indirectly, any CPU device that needs to reset the I/O subsystem. Local bus cards generally don't use this, though it can be used to reset I/O devices or hold the main system in reset if necessary.

/FPURST

This active input is driven by the system reset logic to indicate the full CPU register reset condition.

/CPURST

This open-collector signal is driven by the system reset logic to indicate a full CPU register reset condition to the CPU, and can be driven by the CPU to cause an I/O reset in the rest of the system.

3.3 68030 Signals

All of these signals are directly connected to the 68030, and more information on them is available in the 68030 User's Manual. Most of these must be driven or sampled by any local slot DMA device.

Address Bus ($A_{31}..A_0$)

The 32 bit processor address bus, driven by the bus master, tristated by inactive masters.

Data Bus ($D_{31}..D_0$)

The 32 bit processor data bus, driven by the bus master for writes, the slave for reads. This is tristated when outside of a bus cycle (/AS is negated).

Function Codes ($FC_2..FC_0$)

An address bus extension, driven by the bus master, tristated by inactive masters. Most slaves respond only to $FC_0 \oplus FC_1$.

Bus Size ($SIZ_1..SIZ_0$)

Data bus size request, driven by the bus master, tristated by inactive masters.

Cycle Strobes (/AS, /DS)

/AS indicated the start of a bus cycle and valid addresses, /DS indicates valid data for write cycles. Both are driven by the bus master, tristated by inactive bus masters.

Read Indicator (R/W)

Driven by this bus master, this tristated signal is high to indicate a read, low to indicate a write.

Read-Modify-Write Cycle (/RMC)

This line is asserted by the bus master to effect a bus lock; while its active, no bus arbitration takes place, and shared memory coprocessors (except Agnus) stay out of the memory involved in the transaction.

Other Strobes (/ECS, /OCS, /DBEN)

These are additional 68030 stobes that aren't used by the A3000, and therefore, don't have to be driven by a local bus master. They are provided here for the possible use of any slave device that wants them; see the 68030 User's Manual for signal details.

Burst Control (/CBREQ, /CBACK)

The local bus slave drives /CBREQ to indicate that its capable of supporting a burst cycle. The local bus master responds with /CBACK to indicate that it can run a burst cycle.

Cache Control (/CIIN, /CIOUT)

The bus slave drives the /CIIN line to indicate that the currently addressed location is uncachable. The bus master drives /CIOUT to indicate that the current location is uncachable, based on MMU tables as well as /CIIN.

Cycle Termination (/STERM, /DSACK$_1$, /DSACK$_0$, /AVEC).

The bus slave drives either /STERM or one or more /DTACKs to normally terminate a cycles. The combination of DSACK lines indicates the bus port size; /STERM can only be generated by 32-bit-port slaves. The /AVEC line is driven by local bus logic to terminate an interrupt asknowledge cycle with an autovector rather than a device-supplied vector.

Interrupts (/IPL$_2$-IPL$_0$)

Encoded interrupt inputs. These are generally serviced only by the primary bus master, though other schemes are possible with the proper software support. The are inputs; they can't ever be driven by a slave or a master.

Interrupt Pending (/IPEND)

Driven by the 68030 to indicate that there's a pending interrupt to be serviced. A secondary bus master may use this as an indication to let the 68030 back onto the local bus, if the 68030 is handling the interrupts.

Exceptions (/HALT, /BERR)

/HALT driven alone causes the CPU to stop; generally this is used by single-stepping emulators. /HALT driven with /RESET indicates a full 68000 style reset, and is considered archaic on the A3000 local bus. /BERR driven alone indicates some kind of bus error, generally a bus collision or timeout. /BERR and /HALT driven together indicate a bus retry.

3.4 Bus Arbitration Signals

These are the signals used to arbitrate the local bus in the ways previously described, supporting both primary and secondary bus masters.

Bus Requests (/BR, /CBR)

The both the /BR and /CBR line cause the bus to be requested from the 68030. The /BR line is for secondary bus masters, and it is a time multiplexed open collector line shared with a similar /BR output from the Buster chip. The /CBR line causes a request to go to the 68030, and once the primary arbitration is completed, the new primary master on the local bus slot must deal with incoming /BRs from Buster.

SCSI Bus Request (/SBR)

This line allows the local bus card to monitor when the SCSI devices wants the bus; this is primarily used as an indicator to secondary masters to give up the local bus.

Bus Grants (/BG, /BG30)

The /BG line is the main local bus grant signal. It is normally generated by the 68030, but when a primary master takes over, this line will tristate, allowing the primary master to drive /BG in response to an incoming /BR. The /BG30 line is the bus grant line coming directly from the 68030 chip.

Bus Grant Acknowledges (/BGACK, /BOSS)

The /BGACK signal is the main bus grant acknowledge, shared by Buster and the DMAC. Secondary masters drive /BGACK to acquire the local bus. The /BOSS signal is a private /BGACK-equivalent to the 68030, used by a primary bus master to acquire the bus from the 68030.

Bus Clear Request (/EBCLR)

This is a signal from the bus arbiter, indicating that some other bus master wants the local bus. This is generally used by a secondary bus master as an indicator of when to get off the bus.

3.5 Other Local Bus Signals

Local Slot Memory Decode (/RAMSLOT)

This is an address based chip select for the region of memory allocated to the local bus slot, $08000000-$0fffffff.

Emulator Mode (/EMUL)

This signal can be driven by local bus slot emulator devices to pull the /CDIS and /MMUDIS lines on the 68030, thereby disabling the cache and MMU for debugging purposes.

Cycle Wait (/WAIT)

This line is asserted by a bus monitoring device, such as a cache, to hold off cycle start by either the memory controller (RAMSEY) or expansion bus controller (BUSTER). This gives the device time to determine if it owns that address, retry the cycle, or anything else necessary to support cache and similar kinds of devices.

FPU Chip Select (/FPUCS)

This is a decode for the Coprocessor Device 1, the FPU, generated by the Gary Chip.

3.6 Clocks

This section details the A3000 system clocks available at the local bus slot, the clocking alternatives available to a local bus device, and various clock control lines to facilitate this control.

System Clocks (CPUCLKA, CLK90)

These are the main A3000 system clocks. CPUCLKA is a 16MHz or 25MHz clock, depending on the system configuration, and is the main system, CPU, and FPU clock. CLK90 is CPUCLK shifted 90°.

External Clocks (EXTCLK, EXT90)

These clocks can be driven to replace the on-board clocks to the main system. Some jumpers, described in the appendix, can be arranged to permit the use of these clocks.

3.7 New Amiga 3000T Signals

These signals were added to the A3000T Local Bus Slot. They are also available on the A3000+ Local Bus Slot.

System Clock Steal (DIS_CLKS)

This implements an alternate clock replacement method. When the DIS_CLKS line is driven high, the replacement clocks can be driven onto the local bus. This eliminates the need for any jumper adjustments to be made on the motherboard when a clock sourcing board is installed.

Replacement Clocks (ECPUCLKA, ECPUCLKB, ECLK90, ECLK90A)

These are replacement main system clocks, and their 90° counterparts, that can be directly driven onto the local bus when DIS_CLKS is asserted.

CPU Clock Steal (DIS_CLK30, ECLK30)

This allows the 68030 clock to be driven from the local bus, again for skew reduction or other such tricks. When DIS_CLK30 is asserted, ECLK30 can be driven by local bus card logic.

3.8 New Amiga 3000+ Signals

Several signals were added to the A3000+ Local Bus Slot. Most of these are all no-connects on the A3000 and A3000T systems.

Slot Type Sense (/SENSEA3P)

This signal allows a card to sense whether it's installed in an A3000+ or an A3000/A3000T system. The card should connect a 1K resistor to this line. On an A3000 or A3000T, the line will be high, on an A3000+, it will be low.

Interrupt Out (/INT6)

This is the shared level six interrupt line. It can be driven by a Local Bus Slot device to interrupt the host CPU.

Coprocessor Signals (/CI2P, /PI2C)

These are actually two general purpose lines from the A3000+ Coprocessor Control Register. The /CI2P line is earmarked for a signal from the Coprocessor to the host processor, and the /PI2C line is earmarked for a signal from the host to the Coprocessor. In reality, the lines can be used for any private communications or I/O protocol desired.

APPENDICES

"He saw that the hands that build
could also pull down...."

-U2

A.1 Local Bus Connector Pinout

These signals are described in the text. A more complete description of the standard 68030 inputs and outputs is available from the 68030 User's Manual.

The Local Bus Connector is a two piece type, 200 pin, high density connector. It is made by KEL. Be very careful with connection direction and pinout when doing board layouts.

Signals marked with an asterisk (*) are A3000T and A3000+ signals. Signals marked with a plus (+) are A3000+ signals.

Pin number	Signal name
1	/DSACK$_1$
2	GND
3	GND
4	/HALT
5	R/W
6	GND
7	GND
8	/BGACK
9	/SBR
10	GND
11	GND
12	/AVEC
13	EXT90
14	VCC
15	VCC
16	/RAMSLOT
17	/BOSS
18	VCC
19	VCC
20	FC$_0$
21	/STERM
22	VCC
23	VCC
24	FC$_1$
25	/BR
26	VCC
27	VCC
28	/CBACK
29	/BERR
30	DIS_CLKS*
31	/EMUL
32	/CBREQ
33	A$_8$
34	ECLK30*
35	GND
36	A$_0$
37	A$_9$
38	GND
39	GND
40	A$_1$
41	A$_{10}$
42	ECLK90$_A$*
43	/INT6+
44	A$_2$
45	A$_{11}$
46	ECLK90
47	GND
48	A$_3$
49	A$_{12}$
50	GND
51	GND
52	A$_4$
53	A$_{13}$
54	ECPUCLK$_B$*
55	/WAIT
56	A$_5$
57	A$_{14}$
58	ECPUCLK$_A$*
59	GND
60	A$_6$
61	A$_{15}$
62	GND
63	GND
64	A$_7$
65	A$_{16}$
66	reserved
67	/CI2P+
68	A$_{24}$
69	A$_{17}$
70	reserved
71	GND
72	A$_{25}$
73	A$_{18}$
74	GND
75	GND
76	A$_{26}$
77	A$_{19}$
78	DIS_CLK30*
79	/PI2C+
80	A$_{27}$
81	A$_{20}$
82	reserved
83	GND
84	A$_{28}$
85	A$_{21}$
86	GND

Pin number	Signal name
87	GND
88	A$_{29}$
89	A$_{22}$
90	reserved
91	/DSACK$_0$
92	A$_{30}$
93	A$_{23}$
94	VCC
95	VCC
96	A$_{31}$
97	/DS
98	VCC
99	VCC
100	/ECS
101	/CIOUT
102	VCC
103	VCC
104	/DBEN
105	/BG
106	VCC
107	VCC
108	/RMC
109	/CPURST
110	/FPURST
111	reserved
112	EXTCPU
113	/EBCLR
114	/SENSEZ3P$^+$
115	GND
116	/IPEND
117	/RESET
118	GND
119	GND
120	/IPL$_0$
121	SIZ$_0$
122	GND
123	GND
124	/IPL$_1$
125	FC$_2$
126	CLK90
127	reserved
128	/IPL$_2$
129	SIZ$_1$
130	GND
131	GND
132	/CIIN
133	/AS
134	/FPUCS
135	CPUCLK$_A$
136	/OCS
137	D$_{31}$
138	GND
139	GND
140	D$_{15}$
141	D$_{30}$
142	GND
143	GND
144	D$_{14}$
145	D$_{29}$
146	reserved
147	/CBR
148	D$_{13}$
149	D$_{28}$
150	reserved
151	GND
152	D$_{12}$
153	D$_{27}$
154	GND
155	GND
156	D$_{11}$
157	D$_{26}$
158	reserved
159	/BG30
160	D$_{10}$
161	D$_{25}$
162	reserved
163	GND
164	D$_9$
165	D$_{24}$
166	GND
167	GND
168	D$_8$
169	D$_{16}$
170	reserved
171	reserved
172	D$_0$

173	D_{17}
174	VCC
175	VCC
176	D_{1}
177	D_{18}
178	VCC
179	VCC
180	D_{2}
181	D_{19}
182	VCC
183	VCC
184	D_{3}
185	D_{20}
186	VCC
187	VCC
188	D_{4}
189	D_{21}
190	GND
191	GND
192	D_{5}
193	D_{22}
194	GND
195	GND
196	D_{6}
197	D_{23}
198	GND
199	GND
200	D_{7}

A.2 CPU/Clock Jumpers

There are various strip-post jumpers (links) on the A3000 family motherboards, which control clock speed, sourcing, and other local slot related features.

A.2.1 A3000 Jumpers

These are the A3000 clock configuration jumpers. The A3000 can be preconfigured for either 25MHz or 16MHz operation.

J100 CLK90 Delay Jumper
This jumper has three positions. In position 1-2, it sets up CLK90 for 25MHz operation. In position 2-3, CLK90 is set up for 16MHz operation. In position 3-4, the EXT90 line drives CLK90, rather than the on-board clock logic.

J102 Board Clock
This jumper has two positions. In position 1-2, the source for CPUCLKA and CPUCLKB must be EXTCPU. In position 2-3, it shorts the 68030 and local bus clock sources, such that CPUCLKA, CPUCLKB, and CLK30 all derive from the same source, which is either EXTCPU or the clock based on the local logic, depending on the state of J104.

J103 FPU Chip Select Jumper
This jumper has three positions. With the shunt on pins 1-2, it disables the on-board FPU. With the shunt in position 2-3, it causes the 68030 to get F-line traps instead of the FPU selection, and in position 3-4 it enables the on-board FPU.

J104 CPU Clock
This jumper has two positions. In position 1-2, the source for CLK30 is derived from the on-board clock generator. In position 2-3, the source for CLK30 must come from EXTCLK by way of J102.

A.2.2 A3000T Jumpers

These are the A3000T clock configuration jumpers. The A3000T can be preconfigured for either 25MHz or 16MHz operation.

J100 CLK90 Delay Jumper
Identical to A3000 J100.

J102 Board Clock
Identical to A3000 J102.

J103 FPU Chip Select Jumper
Identical to A3000 J103.

J104 CPU Clock
Identical to A3000 J104.

J105 System Clock Disable
In the 2-3 position, this jumper allows the DIS_CLKS line from the Local Bus Slot to operate normally. In the 1-2 position, it forces DIS_CLKS high (asserted), disabling on-board system clock generation.

J106 CLK90 Short Jumper
In the 2-3 position, there are two versions of CLK90. The system normally runs this way. In the 1-2 position, the two CLK90 lines are shorted together, to allow a single line to source both clocks.

J107 68030 Clock Disable
In the 2-3 position, this jumper allows the DIS_CLK30 line from the Local Bus Slot to operate normally. In the 1-2 position, it forces DIS_CLK30 high (asserted), disabling on-board 68030 clock generation.

A.2.3 A3000+ Jumpers

These are the A3000+ clock configuration jumpers. The A3000+ runs only at 25MHz. Rather than the simple buffer/jumper system of the other A3000 systems, the A3000+ uses a PAL as an intelligent clock generator. This allows it to run clock selectors as logic levels, rather than jumpered clocks, in most cases. This intelligent logic also generates the 50MHz clock for the DSP. This clock will be automatically generated from external clocks rather than the on-board 50MHz clock when an accelerator board is sourcing clocks. Any accelerator setup that removes the clock generator PAL must provide similar functionality on its own.

J100 CLK90 Delay Jumper
This jumper is similar in function to A3000 J100. In position 1-2, it sets up CLK90 for 25MHz operation. In position 2-3, the EXT90 line drives CLK90, rather than the onboard clock logic. Unlike A3000 J100, this jumper is a logic-level switch to the A3000+ clock generator, no clock lines are routed though it.

J102 Board Clock
This jumper is identical in function to A3000 J102, though its a logic-level switch to the A3000+ clock generator; no clock lines are routed through it.

J103 FPU Chip Select Jumper
This jumper is similar in function to A3000 J103. With the shunt on pins 1-2, it enables the on-board FPU. With the shunt in position 2-3, it disables the on-board FPU.

J104 CPU Clock
This jumper is identical in function to A3000 J104, though it's a logic-level switch to the A3000+ clock generator; no clock lines are routed through it.

J105 System Clock Disable
Identical to A3000T J105.

J107 68030 Clock Disable
Identical to A3000T J107.

A.3 References

Motorola, *MC68030 Enhanced 32-Bit Microprocessor User's Manual*, second edition, Motorola Inc. number MC68030UM/AD REV1.

Commodore-Amiga, *The Zorro III Expansion Bus Specification.*

Commodore-Amiga, *The A500/A2000 Technical Reference Manual.*

Programming the A2410 Graphics Card

by Allan Havemose

The Commodore Amiga A2410 high resolution graphics card is designed around the Texas Instruments TMS34010 graphics chip. TMS34010 is a general purpose processor optimized for graphics operations. The A2410 card is controlled by a graphics software system called TIGA (Texas Instruments Graphics Architecture).

This talk covers the following subjects:

- Overview of A2410 hardware.
- Overview of TIGA software.
- Overview of TIGA functions.
- Programming the A2410 with TIGA.

Overview of A2410 Hardware

Function
- Provides the Amiga with a TI TMS34010 graphics coprocessor running at 50 Mhz, to display high-resolution graphics with up to 1024 x 1024 pixels.
- Supports Amiga Autoconfig.

Card Type
- Full size Amiga bus (Zorro II).

Memory
- 2 MB on-board memory (1 megabit, 100ns, ZIP package VRAMs):
- 1 MB video frame buffer
- 0.75 MB program storage
- 0.25 MB overlay bitplanes (two at 0.125 MB each)

Data
- Processor speed of 50 Mhz.
- 16 bit data transfer, programmed transfer or DMA.

Video Display
- 1024 x 1024 maximum display resolution.
- Simultaneous display of 256+3 colors from a palette of 16.8 million.
- Eight bit deep pixels plus two overlay planes.
- Programmable synchronization timing values.
- Software switching between two on-board pixel clock sources.
- Pixel clock sources can be replaced to support custom display resolutions.
- RS343 compatible video with synchronization on green and/or separate TTL synchronization.

Interface Type
- Standard host interface (Zorro II).
- DMA interface (byte-swapped or non-byte-swapped).
- DB15, VGA-style connector.

Power Requirements
- Supplied by the Amiga.

Compatible Monitors
- Monitors capable of 1024 x 784 up to 1024 x 1024 resolution, or analog RGB multisync monitors for lower resolutions.

Overview of TIGA Software

The information in the section is taken from chapter 1 of the TIGA-340 Interface User's Guide.

TIGA, Texas Instruments Graphics Architecture, is a software interface that standardizes communications between application software and TMS340 family-based graphics hardware in personal computers. TIGA divides tasks between the TMS340 processor and the host processor to improve application performance.

The TIGA interface standard simplifies the development of portable applications to the diverse range of TMS340-based systems. TIGA can be extended so that software developers can customize TIGA-340 to a specific application and so that hardware developers can provide a simple interface to specific target features.

TIGA contains a low-level communication interface designed so that other standards can run through the interface with no performance penalty. Essentially, TIGA replaces custom communications routines in other software interfaces with a single standard set of host-to-TMS340 communication routines.

Features

Applications run faster - TIGA-340 provides the application writer with a dual-processor environment. This enables the tasks in the application to be run in parallel by partitioning them between processors. The TIGA-340 interface is optimized to provide high-speed communications between the host and the TMS340 family processors and to minimize the overhead in processing TIGA commands.

Easy to use - TIGA-340 provides applications with a base set of graphics primitives, with all the support required for the graphics subsystem.

Extensible - Where an application requires graphics functions that are not available in the TIGA base set of primitives, that application writer can develop user-extended primitives using TMS340 C, assembly language, or a mixture of the two. These user-extended primitives can be downloaded at run time during the application initialization.

Hardware independent - Inquiry functions are provided that enable the application to determine the resolution, pixel size, etc., of the graphics subsystem and to adapt itself to the board on which it runs.

Overview of TIGA Functions

The following is a list of TIGA functions by functional group. For a complete description of each function, see the *TIGA-340 Interface User's Guide.*

Graphics System Initialization Functions

cd_is_alive	Return if TIGA is available	Core
function_implemented	Return if a function is implemented	Core
get_config	Return board configuration	Core
get_modeinfo	Return board configuration	Core
get_videomode	Return current emulation mode	Core
gsp_execute	Execute a COFF program	Core
install_primitives	Install extended primitives	Core
set_config	Set graphics configuration	Core
set_videomode	Set emulation mode	Core
synchronize	Make host wait for GSP to idle	Core

Clear Functions

clear_frame_buffer	Clear entire frame buffer	Core
clear_page	Clear current drawing page	Core
clear_screen	Clear screen	Core

Graphics Attribute Control Functions

cpw	Compare point to window	Core
get_colors	Return foreground/background colors	Core
get_env	Return current environment structure	Core
get_pmask	Return color plane mask	Core
get_ppop	Return pixel processing operation	Core
get_transp	Return transparancy mode	Core
get_windowing	Return windowing mode	Core
set_bcolor	Set background color	Core
set_clip_rect	Set clipping rectangle	Core
set_colors	Set foreground/background colors	Core
set_draw_origin	Set drawing origin	Ext
set_fcolor	Set foreground color	Core
set_patn	Set current pattern description	Ext
set_pensize	Set current pensize	Ext
set_pmask	Set color plane mask	Core
set_ppop	Set pixel processing operation	Core
set_windowing	Set windowing mode	Core
transp_off	Disable pixel transparency	Core
transp_on	Enable pixel transparency	Core

Palette Functions

get_nearest_color	Return nearest color in palette	Core
get_palet	Return an entire palette	Core
get_palet_entry	Return a palette entry	Core
init_palet	Initialize default palette	Core
set_palet	Set an entire palette	Core
get_palet_entry	Set a palette entry	Core

Graphics Output Functions

draw_line	Draw line	Ext
draw_oval	Draw ellipse outline	Ext
draw_ovalarc	Draw ellipse arc	Ext
draw_piearc	Draw ellipse pie slice	Ext
draw_point	Draw single pixel	Ext
draw_polyline	Draw list of lines	Ext
draw_rect	Draw rectangle outline	Ext
fill_convex	Draw solid convex polygon	Ext
fill_oval	Draw solid ellipse	Ext
fill_piearc	Draw solid ellipse pie slice	Ext
fill_polygon	Draw solid polygon	Ext
fill_rect	Draw solid rectangle	Ext
frame_oval	Draw oval border	Ext
frame_rect	Draw rectangular border	Ext
patnfill_convex	Draw patterned convex polygon	Ext
patnfill_oval	Draw patterned ellipse	Ext
patnfill_piearc	Draw patterned pie slice	Ext
patnfill_polygon	Draw patterned polygon	Ext
patnfill_rect	Draw patterned rectangle	Ext
patnframe_oval	Draw patterned oval border	Ext
patnframe_rect	Draw patterned rectangular border	Ext
patnpen_line	Draw line with pattern and pen	Ext
patnpen_ovalarc	Draw oval arc with pattern and pen	Ext

patnpen_piearc	Draw pie slice with pattern and pen	Ext
patnpen_point	Draw pixel with pattern and pen	Ext
patnpen_polyline	Draw lines with pattern and pen	Ext
pen_line	Draw line with pen	Ext
pen_ovalarc	Draw an oval arc with pen	Ext
pen_piearc	Draw pie slice with pen	Ext
pen_point	Draw point with pen	Ext
pen_polyline	Draw lines with pen	Ext
seed_fill	Fill region with color	Ext
seed_patnfill	Fill region with patterns	Ext
styled_line	Draw styled line	Ext

Poly Drawing Functions

draw_polyline	Draw polyline	Ext
fill_convex	Fill convex polygon	Ext
fill_polygon	Fill polygon	Ext
patnfill_convex	Pattern fill convex polygon	Ext
patnfill_polygon	Pattern fill polygon	Ext
patnpen_polyline	Pattern pen polyline	Ext
pen_polyline	Pen polyline	Ext

Workspace Functions

fill_polygon	Fill polygon	Ext
get_wksp	Return offscreen workspace	Core
patnfill_polygon	Pattern fill polygon	Ext
set_wksp	Set a temporary workspace	Core

Pixel Array Functions

bitblt	Bitblt source array to destination	Ext
set_dstbm	Set destination bitmap	Ext
set_srcbm	Set source bitmap	Ext
swap_bm	Swap source and destination bitmaps	Ext
zoom_rect	Zoom source rectangle	Ext

Text Functions

delete_font	Remove a font from the font table	Ext
get_fontinfo	Return font physical information	Core
get_textattr	Return text rendering attributes	Ext
init_text	Initialize text drawing environment	Core
install_font	Install font into font table	Ext
select_font	Select an installed font for use	Ext
set_textattr	Set text rendering attributes	Ext
text_out	Render an ASCII string	Core
text_width	Return the width of an ASCII string	Ext

Cursor Functions

get_curs_state	Return cursor current state	Core
get_curs_xy	Return cursor position	Core
set_curs_shape	Set cursor shape	Core
set_curs_state	Make cursor visible/invisible	Core
set_curs_xy	Set cursor position	Core

Graphics Utility Functions

get_pixel	Read contents of a pixel	Ext
lmo	Return left-most-one bit number	Core
page_busy	Return status of page flipping	Core
page_flip	Set display and drawing pages	Core
peek_breg	Read from a B-file register	Core
poke_breg	Write to a B-file register	Core
rmo	Return right-most-one bit number	Core
wait_scan	Wait for designated scan-line	Core

Pointer-Based Memory Management Functions

get_offscreen_memory	Return offscreen memory blocks	Core
gsp2gsp	Copy from GSP memory to GSP memory	Core
gsp_calloc	Allocate and clear GSP memory	Core
gsp_free	Deallocate GSP memory	Core
gsp_malloc	Allocate GSP memory	Core
gsp_maxheap	Return largest free block	Core
gsp_minit	Reinitialize GSP memory heap pool	Core
gsp_realloc	Resize allocated block of memory	Core

Communication Functions

field_extract	Extract data from GSP memory	Core
field_insert	Insert data into GSP memory	Core
get_vector	Get address of a TMS340 trap vector	Core
gsp2host	Copy from GSP into host memory	Core
gsp2hostxy	Copy rectangular are from GSP to host	Core
host2gps	Copy from host into GSP memory	Core
host2gspxy	Copy rectangular are from host to GSP	Core
set_vector	Set contents of GSP trap vector	Core

Extensibility Functions

flush_extended	Flush all user extensions	Core
get_isr_priorities	Return interrupt service routine priori.	Core
install_primitives	Install extended drawing primitives	Core
install_rlm	Install relocatable load module	Core
set_interrupt	Set an interrupt handler	Core

A2410 Functions (Amiga specific)

dma_host2gspxy	DMA version of host2gspxy	ULowell
dma16_gsp2gsp	DMA version of gsp2gsp	ULowell
dma_gsp2hostxy	DMA version of gsp2hostxy	Ulowell
draw_planes	Direct drawing to overlays or buffer	ULowell
see_planes	Make overlay planes visible/invisible	ULowell
set_mode_bit	Manually select which oscillator to use	ULowell
set_ovl_color	Set colors used by overlay planes	ULowell
set_spec_reg	Sets lower 3 bits in register	Ulowell
set_swap_bit	Set/reset byte-swap on DMA transfers	ULowell
set_sync_bit	Set sync on green state	ULowell
gsp_halt	Halt the GSP	Core

The following functions are not implemented:

loadcoff	Not of general use to application. Loadcoff is part of LoadTiga
set_timeout	No timer on a2410
set_transp	Only on 34020
cop2gsp	No TI coprocessor on A2410
gsp2cop	No TI coprocessor on A2410
create_alm	ALM's are not supported. Use RLM's
install_alm	-
create_esym	-
flush_esym	-
install_user_error	Not supported

Programming the A2410 with TIGA

This section will give you an overview of how to structure and program a TIGA application. The examples used are very simple, but the general structure of how to open the *A2410.device*, use the TIGA include files, etc. are all covered.

For complete working examples, have a look at the sources included on the DevCon disk.

All communication between your application and the A2410 card is done through the *A2410.device*. *A2410.device* uses two global variables (allocated by your application) when transferring data between the Amiga and the A2410.

```
LONG TIGADATANAME[TIGA_DATA_SIZE];/* data area for macros */
struct IOExtTiga TIGAREQNAME;     /* TIGA IORequest for macros*/
```

All TIGA commands are defined as macros that do some setup and then calls DoIO():

```
#define bitblt(a,b,c,d,e,f)\
((\
  TIGAREQNAME.tiga_Req.io_Command = TIGA_BITBLT,\
  TIGADATANAME[0] = (LONG)(a),\
  TIGADATANAME[1] = (LONG)(b),\
  TIGADATANAME[2] = (LONG)(c),\
  TIGADATANAME[3] = (LONG)(d),\
  TIGADATANAME[4] = (LONG)(e),\
  TIGADATANAME[5] = (LONG)(f),\
  DoIO((struct IORequest *)&TIGAREQNAME)\
)?TIGAREQNAME.tiga_Return:TIGAREQNAME.tiga_Return)
```

The definition of bitblt() shows how TIGADATANAME and TIGAREQNAME are used.

The macros, the io_Command definitions and other TIGA specifics are defined in a set of include files:

A2410/devtiga.h: - Defines structures and constants needed for the device interface.

A2410/dev_macros.h: - Defines macros convenient for accessing the TIGA functions.

A2410/ti_function_nums.h: - Defines the function numbers used by function_implemented() function.

A2410/typedefs.h: - Defines the structures used by TIGA.

Structuring a TIGA Application

A TIGA application could be structured like:

```
void main(int argc, char **argv)
{
    if (TIGA_Init())
    {
        YourFunction();
        TIGA_Close();
    }
}
```

Tiga_Init() and TIGA_Close() are two "wrap" functions that takes care of setting up the IORequest, Open/Close the device, etc. Both functions (and more support functions) can be found in the file *tigainit.c* in the TIGA examples directory.

```
#include "stdio.h"
#include <a2410/typedefs.h>
#include <a2410/devtiga.h>
#include <a2410/ti_function_nums.h>
#ifdef TIGA_MAC
#include <a2410/dev_macros.h>
#else
#include <clib/a2410_protos.h>
#endif
#include <lattice/math.h>

LONG TIGADATANAME[TIGA_DATA_SIZE];
struct IOExtTiga TIGAREQNAME;

BOOL  TIGA_Init(void)
{
int  unit=0;
/* setup the IORequest */
  TIGAREQNAME.tiga_Req.io_Message.mn_Node.ln_Succ= NULL;
  TIGAREQNAME.tiga_Req.io_Message.mn_Node.ln_Pred= NULL;
  TIGAREQNAME.tiga_Req.io_Message.mn_Node.ln_Type= NT_MESSAGE;
  TIGAREQNAME.tiga_Req.io_Message.mn_Node.ln_Pri= 0;
  TIGAREQNAME.tiga_Req.io_Message.mn_Node.ln_Name=NULL;
  TIGAREQNAME.tiga_Req.io_Message.mn_Length=0;
  TIGAREQNAME.tiga_Req.io_Device=NULL;
  TIGAREQNAME.tiga_Req.io_Unit=NULL;
  TIGAREQNAME.tiga_Req.io_Command=CMD_INVALID;
  TIGAREQNAME.tiga_Req.io_Flags=0;
  TIGAREQNAME.tiga_Req.io_Error=0;
  TIGAREQNAME.tiga_Req.io_Actual=0;
  TIGAREQNAME.tiga_Req.io_Length=-1;
  TIGAREQNAME.tiga_Req.io_Data=(APTR)TIGADATANAME;
  TIGAREQNAME.tiga_Req.io_Offset=0;
  TIGAREQNAME.tiga_Return=0;

  TIGAREQNAME.tiga_Req.io_Message.mn_ReplyPort =
       (struct Port *)CreatePort(NULL,0);
  if (TIGAREQNAME.tiga_Req.io_Message.mn_ReplyPort == NULL)
   {
     printf("Can't create the reply port\n");
     return FALSE;
   }

  /* open the device */
  if ((OpenDevice(TIGADEVICENAME,unit,&TIGAREQNAME,LOCK_UNIT) != NULL)
   {
      printf("can't open a2410 device\n");
      DeletePort(TIGAREQNAME.tiga_Req.io_Message.mn_ReplyPort);
      return FALSE;
   }

  /* get TIGA going */
   if  (!set_videomode(TIGA, CLR_SCREEN|INIT))/* PREVIOUS */
    {
       printf("FATAL ERROR - unable to initialize TIGA, aborting...\n");
       CloseDevice(&TIGAREQNAME);
       DeletePort(TIGAREQNAME.tiga_Req.io_Message.mn_ReplyPort);
       return FALSE;
    }
   return TRUE;
}

void TIGA_Close(void)
{
  CloseDevice(&TIGAREQNAME);
  DeletePort(TIGAREQNAME.tiga_Req.io_Message.mn_ReplyPort);
  TIGAREQNAME.tiga_Req.io_Message.mn_ReplyPort = NULL;
}
```

A2410.device support multiple boards. TIGA_Init() only initializes the board with 'unit=0' as shown. A small modification of TIGA_Init() can make it support any unit number. Unit 0 is the first device. Each unit is exclusive access.

Accessing TIGA Functions

You can access the TIGA functions in a couple of different ways:

1. Use the include file *A2410/dev_macros.h* in your source and use the macros it defines. The advantages of this is you do not have to code the setup of an IORequest every time you want a TIGA graphics function; you can code directly from the *TIGA Interface User's Guide* without needing to understand the underlying communication protocols.

2. Use *clib/a2410_protos.h* and *a2410.lib*. *A2410.lib* provides the exact same functionality as the macros (it actually uses the macros), but since it's linked in, there can be a saving in the size of the executable. *clib/a2410_protos.h* just contains prototypes for all TIGA functions. Protypes are good for catching type errors that would go unnoticed when using macros.

3. Both the macros and the link library are synchronous. There is a potential speed advantage to be gained by using asynchronous communication with the device.

Obtaining Configuration Information

Once the *A2410.device* is open, the application can get all the particulars about the current mode of the device and what other modes are available by using the TIGA get_config and get_modeinfo functions, respectively. Get_config takes as its argument the address of the CONFIG structure (see *A2410/typedefs.h*) to fill in.

The CONFIG structure will contain, along with other information, the TIGA revision number, the number of modes that the board supports, the number of the current mode, and a MODEINFO structure. The MODEINFO structure contains all the particulars for the current display mode; such as, display resolution, pixel depth, number of palette entries, size of each palette entry, number of display buffers available (double buffering), etc. An application can use the get_modeinfo function to fill in a MODEINFO structure for each available mode to find the best available display mode for its purposes, then use the set_config function configure the board for that mode. The example program *config* will print to stdout the information in the CONFIG structure and the information in the MODEINFO structure for each available display mode.

Checking Primitive Availability

Not all TIGA functions will be available on all boards that support TIGA. Some functions that might not be implemented on any particular board are:

```
cop2gsp
set_palet
get_palet
set_palet_entry
get_palet_entry
set_transp
gsp2cop
init_palet
```

To determine if a particular TIGA function is implemented on the board the application is using, use the TIGA function, function_implemented. The function_implemented function takes as its argument the function code (defined in *A2410/ti_function_nums.h*) associated with the particular TIGA function of interest, and returns `TRUE` if the function is implemented in this TIGA implementation. The example programs all use function_implemented.

A2410 Overlay Planes

The A2410 provides two one-bit overlay planes in addition to the eight-bit main display buffer. Each plane is manipulated using the same TIGA drawing primitives as the main display buffer, and they can have their visibility set individually. Along with the overlay planes is a three entry color palette separate from the main display buffer color palette. The bits in overlay plane 1 are the least significant bit in the overlay color index.

The following is a descriptions of the functions to control the overlay planes:

draw_plane(plane) - Specifies which overlay plane to draw into.
0 = main display buffer.
1 = overlay plane 1.
2 = overlay plane 2.

When drawing into overlay planes, bit zero of the drawing color is what is written into the plane.

see_planes(planes) - Specifies which overlay planes are used to make the display.
0 = no overlay planes displayed.
1 = overlay plane 1 only, set bits will display overlay color 1.
2 = overlay plane 2 only, set bits will display overlay color 2.
3 = overlay planes 1 and 2, bits in overlay plane 1 are the least significant bit of the overlay color index, for each pixel.

set_ovl_color(ovlindex, red, green, blue) - Sets an overlay palette entry.
Index = the palette entry to set, may be 1, 2, or 3.
red, green, blue = the RGB values for the palette entry. Domain is the same as set_palet_entry.

A2410 Pointer/Cursor

TIGA supports one pointer (or cursor as it is called in TIGA). On A2410 the cursor is implemented using the overlay planes. The default cursor is turned on and of by the same function call:

```
set_cursor_state(1)                  make cursor visible
set_cursor_state(0)                  make cursor invisible
```

It's also possible to install a custom pointer. To do that, the cursor's image must first be defined and then transferred to the GSP. This can be accomplished by the following pseudo code:

```
UCursorData = gsp_malloc(...)
UCursor     = gsp_malloc(...)
host2gsp(cursor_data)
host2gsp(CURSOR)
set_curs_shape(UCursor)
```

For a complete working example, please have a look at *cursor.c* on the DevCon disk.

Extensibility

In contrast to a lot of current graphics standards, TIGA is an extensible interface. Application developers are able to create custom extensions to TIGA using C and TMS340 assembler and then download them to the A2410 to be used along with the standard TIGA functions.

TIGA 1.1 support two types of dynamic load modules:
- Relocatable load modules (RLM)
- Absolute load modules (ALM)

RLM's are produced directly using the TMS340 compiler tools and are in COFF (Common Object File Format) format. These modules contains the necessary relocation entries, so that they can be loaded anywhere in TIGA memory. They may also contain unresolved references to TIGA core or extended primitives. RLM's are installed by using install_rlm().

ALM's are created from RLM's. ALM's are *not* supported by Commodores TIGA implementation. ALM's are basically fixed address versions of RLMs, so that using an ALM requires that the state of the heap in TMS340 memory is the same when the ALM is created as when it is installed. That does not allow for flexible installation of extensions.

Example Programs

On the DevCon disks there are a few example programs. They are all simple in nature but should serve the purpose of showing various features.

boxes: Creates a random palette (if set_palet_entry is implemented) and draws randomly sized boxes at random positions. Use CTRL-C to exit the program.

config: Uses get_config() followed by get_modeinfo() on all possible display modes. Prints the information to stdout.

cursor: Demonstrates how to install and use a TIGA pointer. Opens an Intuition Workbench Window with MOUSEMOVE IDCMP and tracks the Amiga mouse pointer. The program first installs the default pointer and then a custom pointer.

dots: Creates a random palette (if set_palet_entry is implemented) and draws pixels in random color at random positions. Use CTRL-C to exit the program.

lines: Creates a random palette (if set_palet_entry is implemented) and draws a sequence of lines. Use CTRL-C to exit the program.

Note that the wrapper *tigainit.c* is used in all demos.

Hints and Traps

TIGA: When the number of bits per gun isn't 8, the low-order bits are the ones missing. RGB gun values are [0-255].

A2410: The cursor is implemented by using the overlay planes.

TIGA: Before using any of the palette functions, use function_implemented to find out if the palette functions are implemented on the board the application is using.

TIGA: Make no assumptions, use get_config to get the display size.

A2410: Set_config will only succeed if the new mode is supported by the installed oscillators.

A2410: Get_modeinfo will fail for modes not supported by the installed oscillators. Check all modes, not just until first failure.

A2410: Multiple A2410 can be used in a system. Unit number at OpenDevice() call specifies which board. Applications should not assume unit zero is theirs.

A2410: The *A2410.device* only grants exclusive access to each unit.

TIGA: The current release is TIGA 1.1. Make sure that you have the *TIGA 1.1 Users Guide* and not only the newer TIGA 2.0 manual. Many things have been added to TIGA 2.0. Section 2.8.6 in the *TIGA 2.0 Users Guide* list most of the differences between TIGA 2.0 and TIGA 1.1

TIGA 1.1 design: You should be careful when using TIGA 1.1. Unfortunately some TIGA 1.1 takes parameters that are ''wrong'' in the Amiga environment. As an example:

```
    get_palet_entry( long index, char *r, char *g, char *b, char *i)
```

The RGBI values obviously should have been unsigned char. With SAS/C you can set the -cu flag to ensure that all chars are treated as unsigned char.

Using the prototypes together with signed chars can cause the compiler to sign extend, which might cause wrong results.

DMA functions: The DMA functions does not give a performance improvement on a 68030 system. Only use the functions on 68000.

Reference Materials

The following books are recommended for programming the A2410:

TIGA-340 Interface User's Guide, SPVU015A, Texas Instruments, 1989.
The C Programming Language, Kernighan, B. and Ritchie, D., Prentice-Hall, 1978.
Amiga ROM KERNEL Reference Manual: Libraries & Devices, Addison-Wesley
Amiga ROM KERNEL Reference Manual: Includes & Autodocs, Addison-Wesley

Retargetable Graphics

by Chris Green

The current *graphics.library* provides a large set of rendering and display primitives. Unfortunately, this library has always been tied very tightly to the Amiga custom graphics chips, preventing it (and system software which depends on it) from running on incompatible display devices. This is the problem that the Retargetable Graphics (henceforth referred to as RTG) system is intended to solve.

RTG is a backwards compatible upgrade of the Amiga's *graphics.library* which will enable it to render to any graphics device for which compliant graphics device driver software has been written.

Goals

- A high level of compatibility. The most important compatibility issue is to not prevent current software that runs on the Amiga custom chips from running on them in the future. The second compatibility issue is to maximize the number of current well behaved Amiga applications which can run on foreign hardware when present. For applications which are not well behaved enough to run on foreign display hardware, it is important that it be relatively easy for the programmer to upgrade them to be RTG compatible.

- High performance. The speed of graphics operations should be almost totally dependent on the speed of the underlying hardware and the skill of the graphics device driver developer. It is also important that no significant overhead be added for accessing the Amiga native chip set via *graphics.library*.

- Applicability to a large range of graphics devices. This means extending the *graphics.library* interface to handle true-color displays, displays which can't do scrollable viewports, displays with only one page of display memory, displays with smart co-processors, and displays with large amounts of local storage.

- Support for low-level access. One of the things which has distinguished the Amiga is the versatility of the custom chips, and the ability of programmers to use all of their capabilities. RTG does not rule this out. In fact, RTG makes it possible for applications to take advantage of special device capabilities and hopefully provide a fallback when these capabilities are absent.

- A high degree of functionality. The RTG *graphics.library* will provide more functions for the application writer to use than before, thus easing the programmer's burden. Providing new functions also means that these functions can be done in hardware on boards that support it, thus providing additional performance.

RTG Technical Overview

Most graphics calls will work as they do today. Move(), Draw(), SetAPen(), etc. will all work as expected.

The Amiga graphics architecture has up until now only dealt with BitMaps stored in planar form. This remains (with some extensions) the standard bitmap format. RTG introduces the concept of *Natural BitMaps*. A Natural BitMap is an abstract structure representing a two-dimensional array of pixels stored in a device's preferred format. Natural bitmaps will always be allocated by the graphics device driver software, and thus can be any size. Natural BitMaps contain a pointer to the device driver responsible for controlling this bitmap. You may blit from Natural BitMaps to Amiga format BitMaps and vice versa. You may blit from a natural bitmap to another natural bitmap created by the same device. You *may not*, however, blit from a Natural BitMap on one device to one on another device.

Blitting from a planar Amiga style bitmap to a Natural BitMap may be slow. For instance if a planar-to-chunky conversion is involved. For this reason, conversion routines are supplied. A typical usage would be to convert an image from Amiga format to the device's desired format after the application has opened its Screen or Window.

The new AA graphics call AllocBitMap() can be used to create Natural BitMaps.

RTG will fully support true color devices. True color devices and palette mapped devices will appear roughly similar to software. True color devices support a palette (though it is just an indirection table, not a hardware color lookup table), which you can call SetRGB32(), etc. on. Don't expect it to color cycle, though. New versions of SetAPen(), etc. will be provided which allow specification of RGB color values. On a true color device, these will select any color. On a palette mapped device, they will find the closest matching color.

Graphics devices which execute operations asynchronously will be supported. A FlushGraphics() type command will be provided to ensure that all rendering operations have completed.

It will be possible for RTG drivers to be implemented for non-display devices. A virtual buffer approach could be used which would allow *graphics.library* rendering on, for instance, a printer. The virtual buffer approach could also be used to implement a true color driver for HAM mode.

Under RTG, there will be many new flags in the graphics display database which will allow the application to determine the capabilities of a given device. For instance, it will be possible to determine whether or not a device supports viewport scrolling, double buffering, etc.

Graphics device drivers contain a list of jump vectors which point at device specific implementations of the low-level graphics primitives. Many of these entries will be initialized to point at *graphics.library* code which implements these operations in terms of lower-level operations. For instance, the ReadPixelArray() entry could point at *graphics.library* code which does a loop, calling the device's ReadPixel() code. A device driver writer should override this with faster device specific code, but does not have to. Allocating and initializing the vectors in this manner allows new functions to be added, without requiring a re-write for each device driver, as long as these new functions can be implemented in terms of lower level functions.

How to Avoid Breaking under RTG

Doing the following things may break your application under RTG, even when run on the standard amiga chip set. They also might break you under any non-RTG modifications of the *graphics.library*. They are dangerous programming practices. They are things that are likely to break you under the version of graphics for the AA chips as well.

Poking graphics private structures. These include GfxBase, copinit, hardware and software copper lists for viewports, intuition screen structures, and other non-graphics private structures.

Copying ColorMaps. Where Alloc/Free functions exist for data structures, use them! This allows us to extend these structures in the future.

Poking or peeking RastPort structure values. Where Set/Get functions exist, use them!

Not initializing unused fields. Be sure to call InitBitMap() and InitRastPort()!

Using unused structure fields for your own private storage. *Do not do this!*

Do not set the Depth or BytesPerRow field of a bitmap to a negative number! Do not use BitMaps that are greater than 256K pixels wide!

Do not rely on *graphics.library* anomalies.

Do not pass 16 bit values with garbage in the upper word to *graphics.library* routines!

Do not use Sprites without allocating them. Assume that GetSprite() can fail.

The following programming techniques may prevent you from running on foreign display hardware under RTG. They should still work when running on the native Amiga hardware:

Using the blitter to write to display memory. The display memory of a device may not be accessible by the blitter. Using the blitter on your own private bitmap in chip memory, and then using BltBitMap() to copy that to the display will work, though.

Writing or reading to display memory via the BitMap's bm_Planes pointers. On a foreign device, these may not point to memory in the Amiga's memory map. BltBitMap()'ing from this bitmap to one that you have allocated yourself should work OK, though.

User copper lists. Most display devices will not have a copper that is compatible with the Amiga's copper.

Don't use the sc_BitMap field in the Screen structure! Use Screen->RastPort->BitMap instead.

Use new *graphics.library* features. Both the 2.0 *graphics.library* and the AA *graphics.library* have added new features to enable you to do things in a more device independent manner.

Do not rely on the exact pixels output by primitives. The ellipse routine for a foreign display device may not set exactly the same pixels as the native Amiga one, etc.

This is not an exhaustive list. If you can't imagine how what you are doing could possibly work on say, a 34010 card, then chances are that it won't.

◆

AmigaVision Update

By Cathy Godfrey

CATS has recently released a new version of *AmigaVision*--1.70. The following is a list (and short description) of some of its new features:

New Functions

1) EXTPORT. This function allows inter-process communication between *AmigaVision* and an external application without the need for ARexx.

2) MEMORY. To find the amount of memory available in your system, use this function. By passing parameters, you tell the function what kind of memory to report on:
   0 = Total RAM
   1 = Chip RAM
   2 = Fast RAM
   3 = Largest Contiguous Block of memory

3) SELF. Self returns the pathname of the currently running *AmigaVision* flow file. If the parameter for this function is 0, only the filename is given. Otherwise, the entire path is returned.

4) STRAPPEND. The format is STRAPPEND (s1, s2), where s1 and s2 are string variables. This function will append s2 to the end of s1. Although this function is very similar to the STRCAT function, it has one important difference. The STRCAT function assigns the result of the concatenation to the first string (s1). STRAPPEND will not change either string parameter. For example, suppose s1 = "PLAY" and s2 = "GROUND". After STRAPPEND, s1 = "PLAY" and s2 = "GROUND", while after STRCAT, s1 = "PLAYGROUND" and s2 = "GROUND".

Improved SMUS Player

There is now better support for chords, dynamics and tempo in SMUS music files.

Chaining Function

You can use the EXECUTE Icon to chain *AmigaVision* flows together. This is used when you want to divide a large *AVf* file into smaller sections that don't need to be in memory at the same time. The Chaining feature allows the *AmigaVision* author to specify when pieces of code are loaded and executed.

There are two modes, Call Mode and Link Mode.

1) Call Mode. When you use Call Mode, *AmigaVision* treats other *AVf* files as *AmigaVision* subroutines. When the EXECUTE Icon is reached, the specified flow is loaded into memory and run. When this flow terminates, the original flow is continued from the icon immediately after the EXECUTE Icon.

2) Link Mode. When an *AmigaVision* flow is called in Link Mode, it does not automatically return to the calling flow when it is done. Once the Linked flow is terminated, the program is terminated (unless that Linked flow calls another flow file).

New Drivers (Special Notes On The NEC PC-VCR Driver)

AmigaVision 1.70 has several new device drivers: the Carroll Touch touch screen driver, a joystick driver, and three new video drivers - Pioneer 8000 laserdisc, Panasonic OMDR, and the NEC PC-VCR driver.

The PC-VCR is a great tool for those schools or companies that already have educational or training materials on videotape. They can use their existing material with *AmigaVision* without having to spend the money to convert the tape to laserdisc.

The NEC PC-VCR driver is special. An *AmigaVision* flow written for other players will not work the same way on the PC-VCR and vice versa. There are many factors to consider when authoring with a PC-VCR:

1) This player does not support reverse play except for FAST REVERSE mode.

2) PC-VCR will only return the Frame number while in STILL mode. Therefore you cannot write code that starts video playing and then uses a CONDITIONAL WAIT to trigger a sound effect or brush on a specific frame.

3) STILL mode is maintained for only five minutes. This is to prevent damage to the videotape. After five minutes, the machine goes into PLAY mode.

4) The biggest consideration is way that frame number information is represented. The PC-VCR uses an Address:Frame time signature. The Address number specifies the position of the tape in seconds. The Frame number specifies which frame (each second of video contains thirty frames) is being displayed (0 - 29).

AmigaVision supports a five-digit frame number, so it supports the PC-VCR time signature in the following way:

AAAAF, where AAAA is the Address number and F is the Frame number divided by 3.

Therefore, ''07032'' = Address 0703, Frame 6 and ''63129'' = Address 6312, Frame 27.

AmigaVision Player

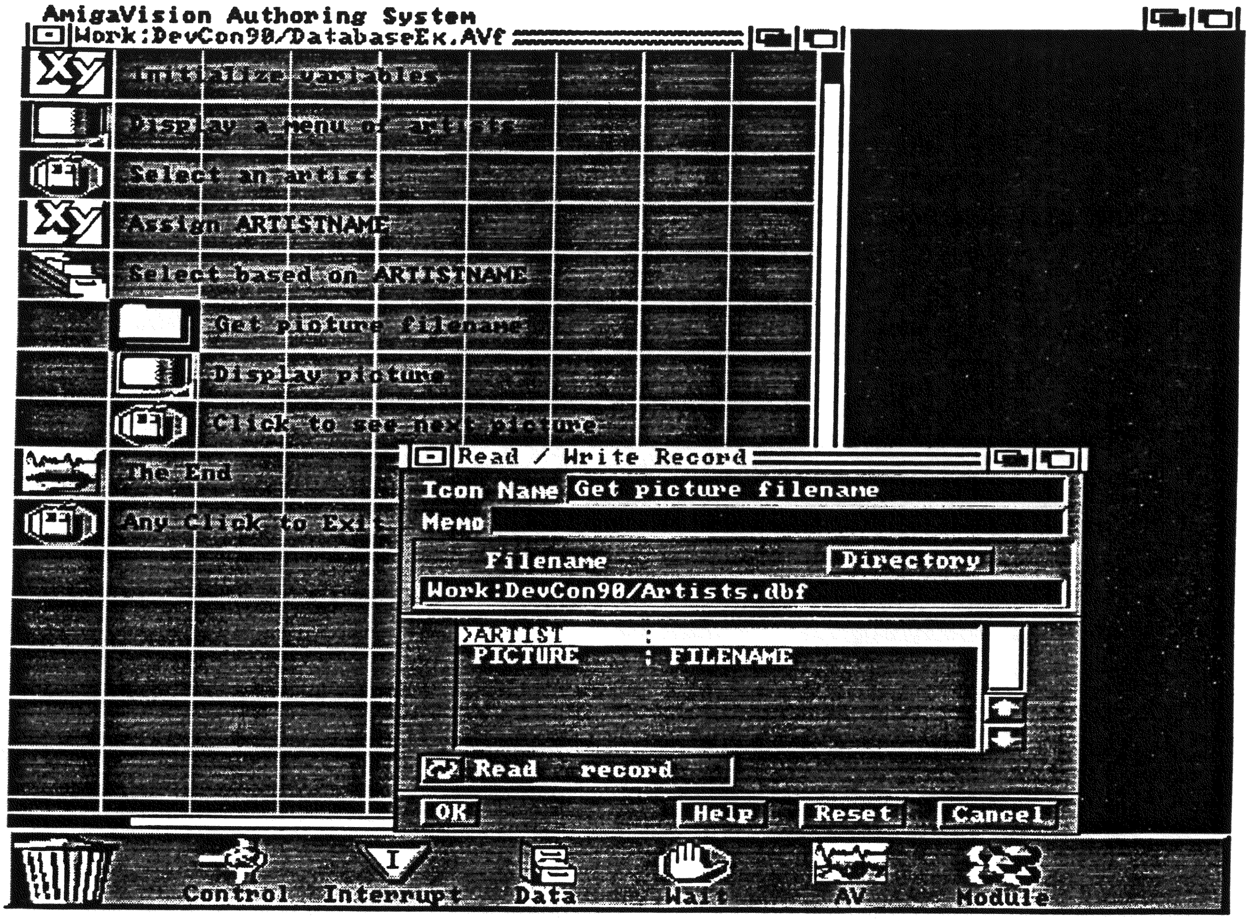
There is now an *AmigaVision* player program. If you decide to license this program from Commodore, you can include it with your own *AmigaVision* applications so that your users don't need to own *AmigaVision* in order to run your software.

Other advantages of the player program are:

1) It supports all new features of version *AmigaVision* 1.70.
2) It will save over 350K of disk space (compared to using *AmigaVision*).
3) It will saves approximately 50K of memory.

Solving Some AmigaVision Problems

How to use *AmigaVision*'s Database


AmigaVision Authoring System
Work:DevCon90/DatabaseEx.AVf
Initialize variables
Display a menu of artists
Select an artist
Assign ARTISTNAME
Select based on ARTISTNAME
Get picture filename
Display picture
Click to see next picture
The End
Any Click to Exi
Read / Write Record
Directory
Read record
OK
Help
Reset
Cancel
Control
Interrupt
Data
Wait
AV
Module

Icon Name Get picture filename

Memo

Filename

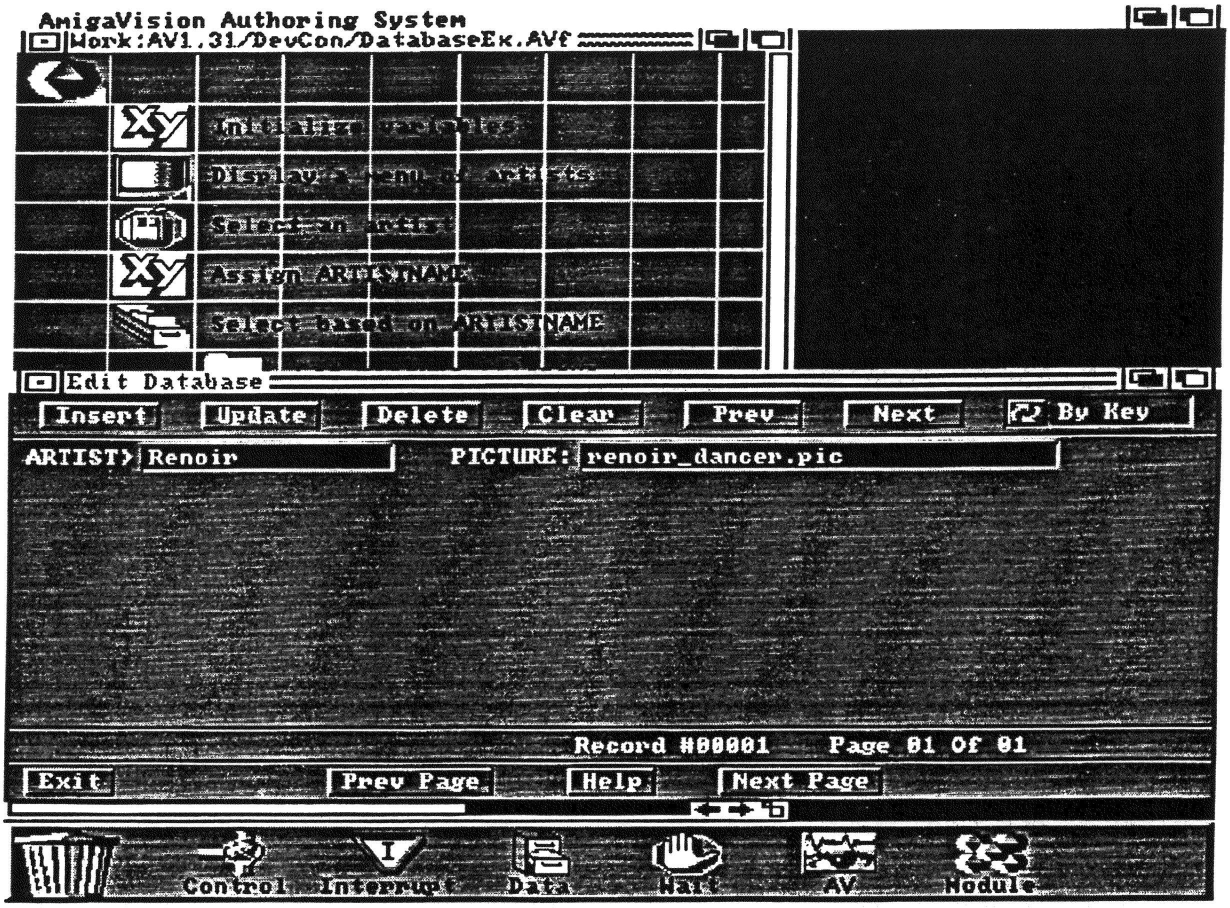
Work:DevCon90/Artists.dbf

ARTIST :

PICTURE : FILENAME

Figure-1 *DBase-1*

Figure Dbase-1 shows an example of a simple *AmigaVision* application which uses the built-in database feature of the authoring system. This example will allow the user to choose an artist and then view painting created by that artist. The list of paintings and its artist is kept in the database, *Artist.dbf*.


AmigaVision Authoring System
Work:AV1.31/DevCon/DatabaseEx.AVf
Initialize variables
Display a menu of artists
Select an artist
Assign ARTISTNAME
Select based on ARTISTNAME
Edit Database
Insert
Update
Delete
Clear
Prev
Next
By Key
Exit
Prev Page
Help
Next Page
Control
Interrupt
Data
Wait
AV
Module

ARTIST> Renoir

PICTURE: renoir_dancer.pic

Record #00001 Page 01 Of 01

Figure-2 *DBase-2*

First, a brief description of the database we will use for this example. *Artist.dbf* is a database created in *AmigaVision* (see Figure Dbase-2). Each record has two fields: ARTIST (which contains the name of the artist) and PICTURE (which contains the filename of a picture created by that artist). In this database there are three records listing paintings by Renoir, three by Monet, and one by Degas.

Now back to the *AmigaVision* flow window... The first VARIABLE Icon initializes two variables, ARTISTNAME and FILENAME. ARTISTNAME will keep track of which artist the user selected and FILENAME will contain the name of the picture file to be displayed.

The next two icons (SCREEN Icon and WAIT MOUSE Icon) set up a menu. The SCREEN Icon displays the menu and the WAIT MOUSE Icon sets up the hit boxes. This menu is a list of the artists that the user has to choose from. When the user clicks on an artist, the VARIABLE Icon will assign the user's response to the variable ARTISTNAME.

The next icon is called the SELECT Icon. It allows you to access specific records within a database. You choose these records by matching one (or more) of the fields with a variable(s). In this example we will match the field ARTIST with our variable, ARTISTNAME. Every time *AmigaVision* finds a record in which the field ARTIST matches the name in the variable ARTISTNAME, it will execute the actions of the children of the SELECT Icon.

The first thing *AmigaVision* will do when it selects a record is to execute the READ/WRITE Icon. This icon handles input to and output from the database. Our READ/WRITE Icon will read the value in the PICTURE field and place it in the variable FILENAME.

The SCREEN Icon will use the variable FILENAME to display the picture create by the chosen artist. Then the application waits until the user clicks on the mouse.

When *AmigaVision* has searched the entire database for matches, it then executes the actions of the SELECT Icon's siblings. Our database example will use a SPEECH Icon to tell the user that they have reached the last picture. A final mouse click will end the application.

The database has many practical applications. This *AmigaVision* example, for instance, could easily be extended to include images stored on a videodisc. If our database had an extra field specifying frame number, we could use the VIDEO Icon to show an image of the painting stored on a videodisc. Any other information within the database (name of painting, year created, etc.) could easily be read from the database and (using the GFX Icon) graphically overlaid on the video image of the painting.

How to Add a Field to an AmigaVision Database

Image you are creating a database in *AmigaVision*. Carefully planning each field type and every field size...entering hundreds of records by hand...checking and then double-checking each line for typos...and then you find that you need to add another field to the database. UGH!

You have a few options.

1) Start all over again (for *AmigaVision* won't let you add a new field to an existing database without deleting all existing information first).
2) Buy and learn how to use an Amiga-based, DBaseIII-compatible program that allows you to add fields to an existing database.
3) Use a quick and easy *AmigaVision* flow to transfer your data from the original database into the new one.

Let's try option 3.

Suppose your original database is *MyData.dbf* and you've forgotten to add a field, AGE. First, start at the CLI, use the copy command and create a duplicate of the original database (we'll call the copy *NewData.dbf*).

Back in *AmigaVision*, load *NewData.dbf* into the Database Editor and DELETE its data. Now you can INSERT the new field, AGE. Click on CREATE to save this database.

In a new Flow window, use the VARIABLES Icon to create one variable for every field that is in *MyData.dbf*.

As a sibling of the VARIABLES Icon place a SELECT Icon. *MyData.dbf* should be specified in its Filename field. This SELECT Icon will have two children, both R/W Icons.

The first R/W Icon should be set to READ RECORD. Place *MyData.dbf* in its Filename field. Match every record in MyData to a variable. Set the second R/W Icon to INSERT RECORD. Place *NewData.dbf* in its Filename field also. Again, match the records in NewData to its corresponding variable (only your newly added field will not have a variable matched to it).

When you Present this flow, all the information from MyData will be read into variables and then transferred to NewData.

Add other icons between the R/W icons to aid in the creation of this new database. For example: Add a GFX Icon to display a key field variable so you can track the progress of you flow. Or add a FORM Icon to solicit input. Then you can enter data into the new field while you're transferring the information.

How to Speed up Hit Boxes

(This article will refer to the *Keyboard.AVf* flow that comes on your *AmigaVision* Tutorial diskettes.)

Have you noticed that there is a slight delay between the time that you display hit boxes on the screen and the time that they are active? This minor delay can become critical if there are many hit boxes on the screen. If these hit boxes could be pre- loaded, it would allow the user to click on the hit boxes as soon as they are displayed.

One way to pre-load hit boxes is to pre-display them...that is, actually display your hit boxes in one icon and then wait for the user to click on them in another. We will use the *Keyboard.AVf* tutorial as an example of how we can speed up hit boxes by moving them to another icon.

The problem with *Keyboard.AVf* is that after the user selects a letter he must endure a delay before he can choose another letter. Letters will be missed if the user clicks too fast. The reason for this wait is that the hit boxes are erased from the screen after each letter is chosen. So before the user can click on a new letter, *AmigaVision* must load all those object on the screen again (over 30 of them).

If we could load in all the hit boxes at once outside the "Accept Letters" loop, then there would be no delay between selecting letters.

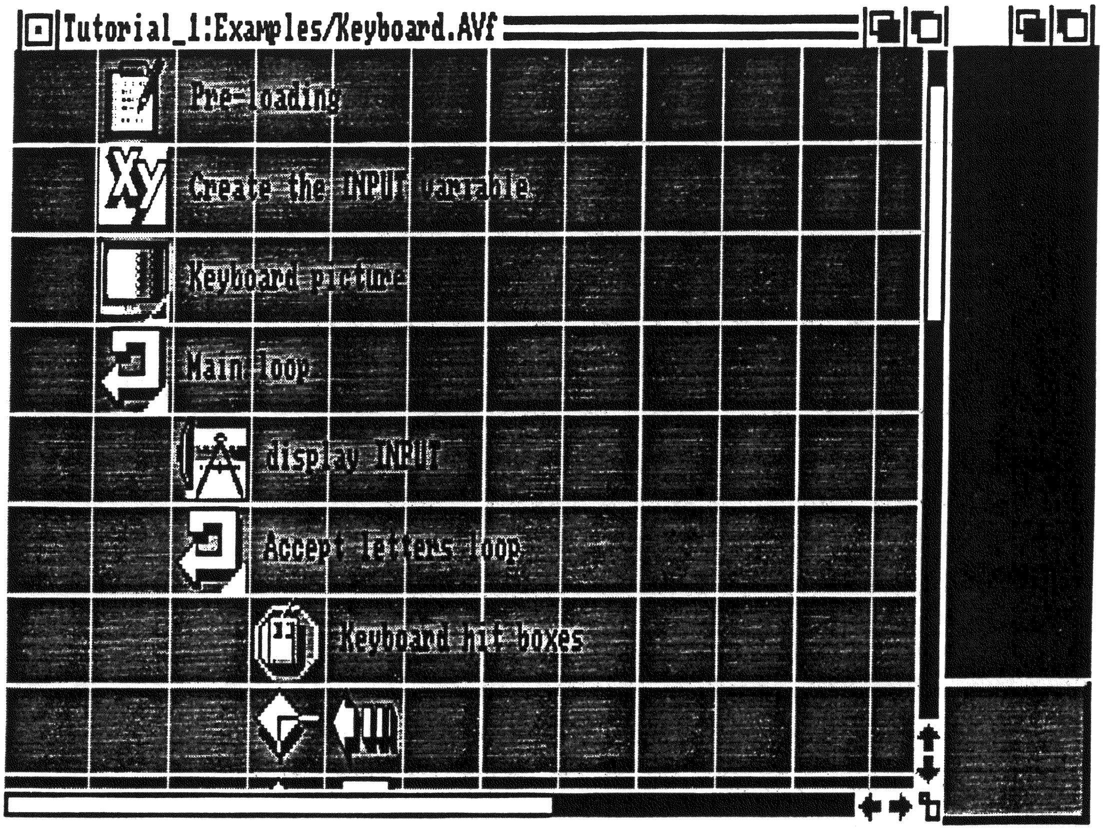

Tutorial_1:Examples/Keyboard.AVf
Pre-loading
Create the INPUT variable
Keyboard picture
Main loop
display INPUT
Accept letters loop
Keyboard hit boxes

Figure-3 *Keyboard*

A very simple way to do this is to first move the GFX icon called "display INPUT" out of the "Accept Letters" loop and make it the first child of the "Main Loop" LOOP Icon (See Figure Keyboard.1). Then transfer the hit boxes from the WAIT MOUSE Icon called "Keyboard hit boxes" into the GFX Icon. The GFX Icon puts the hit boxes on the screen once and then (within the "Accept Letters" loop) the user can click on then without having to wait.

NOTE: After you transfer the hit boxes, make sure that the Background option on the GFX Icon is turned OFF. Otherwise, you graphic objects will not be hit boxes.

If you compare reaction time of the old Keyboard application and the modified one, you will see that the user can click very fast and the modified *Keyboard.AVf* will not "miss" letters, like the original did.

Keep in mind that hit boxes don't necessarily have to be created in Wait Icons and that they don't have to be re-displayed every time you need them. Like pre-loading sound or music into memory before playing it, pre-displaying hit boxes will make your application run quicker and smoother.

Installer Version 1.0 Documentation

by Paul Higginbottom

Contents:

Background

Overview
- **Standard Invocation**
- **Initial Actions**
- **Startup Screens**
- **Installation Actions**

Scripting Language Tutorial
- **Basic Elements**
- **Escape Characters**
- **Symbols**
- **Types of Symbols**
- **Statements**
- **Data Types**
- **Special Features**
- **Miscellaneous**

Installer Language Reference
- **Notes**
- **Statements**
- **Control Statements**
- **Functions**
- **Summary of Parameters**
- **Pre-Defined Variables**

Installer Language Quick Reference
- **Overview**
- **Quick Language Overview**
- **Pre-Defined Variables**
- **Default Help String Variables**
- **Statements**
- **Functions**

Background

Installation of applications from floppy disks onto a hard disk has proven to be a very inconsistent and often frustrating endeavor for most end-users. This has been caused by many factors, some of which are:

1. Many products do not come with any utility or script to install an application on a hard disk.
2. Many products assume a great deal of familiarity with the startup process of the Amiga and applications, including assigns, device names (as opposed to volume names), etc.
3. The installation scripts or utilities included with some products vary widely in their ability to deal with different environments and systems.

About a year ago, Commodore set out to remedy this situation, by developing a standard tool that developers can include with their products, which provides the user with a standard way to install applications. The Installer's features were based on a number of assumptions:

1. Installation requirements vary widely--some need assigns, some need new drawers created, some install pieces in system drawers such as a fonts drawer, a 'product' might be just an upgrade and the installation must check to see which version (if any) they currently have installed, etc.
2. Different users have different levels of comfort and expertise when attempting to install software, and the Installer should be able to accomodate a range of users. Many installation scripts assume a great deal of knowledge, which is very intimidating for a novice.
3. The installer tool must be very flexible internally, but present a consistent pleasant graphical user interface to the user that only shows the user information or prompts that they need to see. The Installer should be resolution, color and font sensitive.
4. Writing scripts to install an application will require some effort, but certainly no more than writing an AmigaDOS shell script equivalent, and the resulting installation procedure will be more friendly, flexible, and much better looking than the latter.
5. Not everyone will be running 2.0 by the time the tool becomes available, so it must run under 1.3 and 2.0.

I. Overview

The Installer is a script driven program, that presents a consistent installation environment to the end user. The user never sees the script. Instead they are presented with simple yes/no choices, and may be asked to specify locations to put things on their system.

To accomodate different user levels, they can choose to run the tool in novice, average or expert modes. Scripts can include help text to explain any choices that the user must make. At each step the user is given the option of aborting the installation.

Standard Invocation

The Installer is normally started up from a Workbench Project icon which has the same name as the script to interpret and has a default tool of Installer. A number of tooltypes are available to modify the operation of the Installer:

APPNAME - Name of the application being installed (appears in the startup screen). This MUST be given.

MINUSER - The minimum possible operation mode of the installation for a script. This will be either NOVICE (all decisions made by Installer), AVERAGE (only important decisions made by user) or EXPERT (user gets to confirm almost all actions). The default is NOVICE.

DEFUSER - Indicates which operation mode button should be initially selected. Same values as MINUSER, with the value of the MINUSER tooltype being the default (which will be NOVICE if MINUSER not defined).

PRETEND - If set to FALSE, indicates that PRETEND mode not available for this script.

LOGFILE - The name of the log file that the Installer should use. The default is "install_log_file".

LOG - In NOVICE mode the default is to create a log file (to disk). If this tooltype is set to FALSE, the creation of a log file in NOVICE mode is disabled.

Although the installer can be started up from the CLI, that is not the recommended mode. CLI invocation is provided mainly for script debugging purposes. The command template is:

```
SCRIPT/k,APPNAME,MINUSER,DEFUSER,LOGFILE,NOLOG/s,NOPRETEND/s
```

Initial Actions

The first thing the installer does is compile the installation script into an internal format that can be easily interpreted. If there are syntax errors in the script, they will be caught during this phase.

Startup Screens

Next, the Installer asks the user what Installation Mode to run in, either NOVICE, AVERAGE or EXPERT. If the user chooses NOVICE, they will not be asked any more questions (although they may be requested to do things). In the other user levels, a second display appears asking the user if he wants to install ''for real'' or ''do a dry run'', and if he wants a transcription of the installation process written to either a file or printer.

Installation Actions

Now the Installer interprets its internal version of the script. Any commands that call for a user interface will cause the Installer to algorithmically generate a display, always including buttons to allow for context sensitive help and aborting the installation.

II. Scripting Language Tutorial

The script langauge of the Installer is based on LISP. It is not difficult to learn, but requires a lot of parentheses. An Installer script can easily be made to look very readable.

Basic elements: The basic elements of the installer language are:

Type	Example
decimal integers	5
hexadecimal integers	$a000
binary integers	%0010010
strings	"Hello" or 'Hello'
symbols	x
comments	; this is a comment
( )	for statement definition
space (or any white space)	delimits symbols

Escape-characters are supported as in the C language:

Escape sequence	Produces
'\n'	newline character
'\r'	return character
'\t'	tab character
'\0'	a NUL character
'\"'	a double-quote
'\\'	a backslash

Symbols: a symbol is any sequence of characters surrounded by spaces that is not a quoted string, an integer or a control character. This means that symbols can have punctuation marks and other special characters in them. The following are all valid symbols:

x
total
this-is-a-symbol
name
@#__#@

Types of symbols: There are three types of symbols:

- user-defined symbols. These are created using the ''set'' function.
- built-in function names. These include things like '+' and '*' as well as textual names such as ''delete'' or ''rename''.
- special symbols. These are variables which are created by the installer before the script actually starts to run, and are used to tell the script certain things about the environment. These symbols always begin with an '@' sign. An example is '@default-dest' which tells you the default directory that was selected by the installer.

Statements: The format of a statement is:

```
(operator <operand1> <operand2> ...)
```

A statement to assign the value ’5’ to the variable ’x’ would be:

```
(set x 5)
```

You can read this as “set x to 5”. Note that the variable ’x’ does not have to be declared -- it is created by this statement.

Note that there is no difference between operators and functions -- the function ’set’ and the arithmetic operator ’+’ are both used exactly the same way.

Combining statements: A statement can be used as the operand to another statement as follows:

```
(set var (+ 3 5))
```

In this case, the statement ’(+ 3 5)’ is evaluated first, and the result is 8. You can think of this as having the ’(+ 3 5)’ part being replaced by an 8. So now we are left with:

```
(set var 8)
```

-- which is the same form as the first example.

Note that the ’(+ 3 5)’ part actually produced a value: “8”. This is called the “result” of the statement. Many statements return results, even some that might surprise you (such as “set” and “if”).

Data Types: All data types in the installer are dynamic, that is to say the type of a variable is determined by the data that is in it. So if you assign the string “Hello, World” to the variable ’x’, then ’x’ will be of type STRING. Later you can assign an integer to ’x’ and x will be of type INTEGER. When using variables in expressions, the interpreter will attempt to convert to the proper type if possible.

Special forms: There are two exceptions to the form of a statement. The first type is used for string substitution: If the first item in parentheses is a text string rather than a function name, the result of that clause is another string that is created by taking the original string and performing a “printf”-like formatting operation on it, using the other arguments of the statement as parameters to the formatting operation.

Thus the statement:

```
("My name is %s and I am %ld years old" "Mary" 5)
```

Becomes:

"My name is Mary and I am 5 years old"

Note since the formatting operation uses the ROM “RawDoFmt” routine, you must always say "%ld" rather than "%d" (The interpreter always passes numeric quantities as longwords).

The second type of exception occurs if the elements in parentheses are themselves statements in parentheses. In this case, the interpreter assumes that all the elements are statements to be executed sequentially.

For example, this statement sets the value of three different variables, ''var1'', ''var2'' and ''var3'':

```
((set var1 5) (set var2 6) (set var3 7))
```

What this feature does is allow the language to have a block structure, where an ''if'' statement can have multiple statements in it's ''then'' or ''else'' clause. Note that the result of this statement will be the result of the last statement in the sequence.

Complex statements: Here is an example of how statements in the script language can be combined into complex expressions. We will start with an ''if'' statement. The basic format of an ''if'' statement is:

```
(if <condition> <then-statement> [<else-statement>])
```

The condition should be a statement which returns a value. Then ''then'' and optional ''else'' parts should be statements. Note that if the ''then'' or ''else'' statements produce a result, then the ''if'' statement will also have this result.

Our first example is a rather strange one: Using an ''if'' statement to simulate a boolean ''not'' operator. (Note that there are easier ways in the script language to do this).

```
(set flag 0)        ; set a flag to FALSE

(set flag (if flag 0 1)); a Boolean NOT
```

Basically, the ''if'' statement tests the variable ''flag''. If flag is non-zero, it produces the value ''0''. Otherwise, the result is ''1''. In either case, ''flag'' is set to the result of the ''if'' statement.

Now, let's plug some real statements into our ''if'' statement.

```
(if flag            ; conditional test
      (message "'flag' was non-zero\n"); "then" clause.
      (message "'flag' was zero\n"); "else" clause.
)                   ; closing parenthesis
```

Note the style of the indenting. This makes for an easier to read program.

Now, we'll add a real condition. The "=" operator tests for equality of two items.

```
(if (= a 2)         ; conditional test
      (message "a is 2\n"); "then" clause
      (message "a is not 2\n"); "else" clause
)                   ; closing parenthesis
```

Finally, just to make things interesting, we'll make the "else" clause a compound statement.

```
(if (= a 2)         ; conditional test
      (message "a is 2\n"); "then" clause
      (     (message "a is not 2\n"); "else" compound stmt
            (set a 2)
            (message "but it is now!\n")
      )             ; end of compound statement
)                   ; end of if
```

Special Features

When the Installer first starts up, it attempts to determine the ''best'' place to install the application. Any volume named ''WORK:'' is given preference, as this is the standard way that an Amiga comes configured from Commodore.

There are two keyboard shortcuts. Whenever there is a ''Help'' button active, pressing the HELP key will also bring up the help display. Whenever there is an "Abort" button active, pressing ESC brings up the abort requester. Also, whenever the installer is "busy", pressing ESC brings up the abort requester -- there is text is the title bar to that effect.

If an application must have assigns or other actions performed during system boot, the Installer will add these to a file named *S:user-startup*. The installer will then add the lines

```
if exists S:user-startup
execute S:user-startup
endif
```

to the user's *startup-sequence*. The Installer will attempt to determine the boot volume of the system when looking for the *startup-sequence* and can handle any AmigaDOS scripts executed from *startup-sequence* (up to 10 levels of nesting).

The Installer can create an assign to just a device, volume or logical assignment. This comes in handy when you want to update an application which comes on a volume named "MyApp:", but the installed version is in a directory with the logical assign "MyApp:"!

The Installer always copies files in CLONE mode, meaning all the protection bits, filenotes and file dates are preserved. When copying files the Installer gives a "fuelgauge" readout of the progress of the copy.

The Installer can find the version number of any executable file that has either a RomTag with an ID string (such as libraries and devices) or has a version string conforming to that given in the 1990 DevCon notes. The Installer can also checksum files. A seperate utility named "instsum" is provided to determine a file's checksum for use with this feature.

Miscellaneous

To perform a set of actions on all the contents of a directory matching a pattern you can use the "foreach" operator. To perform a set of actions on an explicit set of files, the following installer statements can be used as a template:

```
(set n 0)
(while (set thisfile (select n "file1" "file2" "file3" ""))
       (      (set n (+ n 1))
              (...  your stuff involving thisfile ...)
       )
)
```

Note that an empty string is considered a FALSE value to any condition operator.

To run an external CLI command which normally requires user input, redirect the input from a file with the needed responces. For example, to format a disk one could combine the statement shown below with a file which contains only a newline character.

```
(run "format <nl_file drive DF0: name ToBeEmpty")
```

III. Installer Language Reference

Notes

When the script exits either by comming to the end or via the ‘‘exit’’ statement, a message will be displayed saying where the application was installed and where the logfile (if any) was written. Note that you must store in ‘‘@default-dest’’ where you actually installed the application (see ‘‘@default-dest’’ below).

A newline character ('\n', 0x0a) will cause a line break when the installer performs word-wrapping.. A hard-space (ALT-space, 0xa0) will prevent a word break when the installer performs word-wrapping. Also, quoted sections will be considered one word for word-wrapping purposes. For example, if the following help text was used:

"‘The disk name \"FrameZapper 2.0\" is needed to complete installation."

Then the text ‘‘FrameZapper 2.0’’ will not have a word break before the ‘‘2’’.

Statements

```
(set <varname> <value> [<varname2> <value2> ...])
```

Set the variable <varname> to the indicated value. If <varname> does not exist it will be created. Set returns the value of the last assignment.

Note: All variables are typeless, and any variable may be used wherever a string could be used. All variables are global.

`(makedir <name> <parameters>)` - Creates a new directory. Parameters:

prompt - tell the user what's going to happen.
help - text of help message
infos - create an icon for directory
confirm - if this option is present, user will be prompted,
    else the directory will be created silently.
safe - make directory even if in PRETEND mode

`(copyfiles <parameters>)` - Copies one or more files from the install disk to a target directory. Each file will be displayed with a checkmark next to the name indicating if the file should be copied or not. Parameters:

prompt, help - as above
source - name of source directory or file
dest - name of destination directory, which is created if it
    doesn't exist
newname - if copying one file only, and file is to be renamed, this is

the new name
choices - a list of files/directories to be copied (optional)
all - all files/directories in the source directory should be copied
pattern - indicates that files/directories from the source dir
matching a pattern should be copied
files - only copy files. By default the installer will match and copy
subdirectories.
infos - switch to copy icons along with other files/directories
fonts - switch to not display ".font" files, yet still copy any that
match a directory that is being copied.
optional - makes it not a fatal error if a file doesn't copy
confirm - if this option is present, user will be prompted to
indicate which files are to be copied, else the files
will be copied silently.
safe - copy files even if in PRETEND mode.

`(copylib <parameters>)` - Copies one file using version checking; i.e., it only overwrites an existing file if the new file has a higher version/revision number. Parameters:

prompt, help - as above
source - name of source directory or file
dest - name of destination directory
newname - if copying one file only, and file is to be renamed,
this is the new name
infos - switch to copy icons along with other files
optional - makes it not a fatal error if a file doesn't copy
confirm - user will be asked to confirm. Note that an EXPERT user
will be able to overwrite a newer file with an older one.
safe - copy the file even if in PRETEND mode

`(startup <appname> <parameters>)` - This command edits the *S:user-startup* file, which is executed by the user's *startup-sequence* (Installer will modify the user's *startup-sequence* if needed, although in a friendly way). The "command" parameter is used to declare AmigaDOS command lines which will be executed. The command lines are grouped by application, using the supplied argument "appname". If there is already an entry in *S:user-startup* for that application, the new command lines will completely replace the old. The command lines for other applications will not be affected. Parameters:

prompt, help - as above
command - used to declare an AmigaDOS command line to be
executed at system startup.

`(tooltype <parameters>)` - Modify an icon's tool type. Normally the new tool type values will be set up in advance by various statements in the install language (i.e. the user does not actually have to type in the tooltype values). For example, you could use an "askchoice" to ask the user what type of screen resolution they want and then format the tooltype string based on their choice. The "tooltype" operation merely asks for a confirmation before actually writing. Parameters:

prompt, help - as above
dest - the icon to be modified
settooltype - the tooltype name and value string.
setdefaulttool - default tool for a project
setstack - set size of stack
noposition - reset to NOICONPOSITION
confirm - if this option is present, the user will be asked for
    confirmation, otherwise the modification proceeds
    silently.
safe - make changes even if in PRETEND mode

`(textfile <parameters>)` - Creates a text file from other textfiles or computed text strings. This can be used to create configuration files, AREXX programs or execute scripts. Parameters:

help, prompt - as above
dest - the name of the text file to be created.
append - a string to be appended to the new text file.
include - a text file to be appended to the new text file.
confirm - if this option is present, the user will be asked for
confirmation, otherwise the writing proceeds silently.
safe - create file even if in PRETEND mode

`(execute <argument> ...)` - Executes an AmigaDOS script with the arguments given. NOTE: Does not ask user for confirmation, however this can be added by using "askchoice" or "askbool". Parameters:

help, prompt - as above
confirm - if this option is present, the user will be asked for
    confirmation, otherwise the execute proceeds silently.
safe - execute script even if in PRETEND mode

`(run <argument> ...)` - Executes a compiled program with the arguments given. NOTE: Does not ask user for confirmation, however this can be added by using "askchoice" or "askbool". Parameters:

help, prompt - as above
confirm - if this option is present, the user will be asked for
    confirmation, otherwise the run proceeds silently.
safe - run program even if in PRETEND mode

`(rexx <argument> ...)` - Executes n ARexx script with the arguments given. NOTE: Does not ask user for confirmation, however this can be added by using "askchoice" or "askbool". If the ARexx server is not active, an error will be generated. Parameters:

help, prompt - as above
confirm - if this option is present, the user will be asked for
confirmation, otherwise the rexx script proceeds silently.
safe - execute script even if in PRETEND mode

`(makeassign <assign> [<path>] (parameters))` - Assigns 'assign' to 'path'. If 'path' is not specified, the assignment is cleared. Parameters:

safe - execute script even if in PRETEND mode
Note: assign must be supplied without a colon; i.e. "ENV" not "ENV:".

`(rename <oldname> <newname> <parameters>)` - Renames a file or directory. If the "disk" parameter is given, then this command relabels the disk named oldname to newname. When relabeling a disk, ONLY include a colon in the oldname. Returns 1 if the rename was successful, 0 if it failed. Parameters:

help, prompt - as above
confirm - if this option is present, the user will be asked for
  confirmation, otherwise the rename proceeds silently.
disk - switch to get rename to relabel a disk.
safe - rename even if in PRETEND mode

`(delete <file> <parameters>)` - Delete a file. Parameters:

help, prompt - as above
confirm - if this option is present, the user will be asked for
  confirmation, otherwise the delete proceeds silently.
safe - delete even if in PRETEND mode

`(protect <file> [<protection-bits>] <parameters>)` - Either gets the protection bits for a file (if second argument not given), or sets them. When setting bits, returns 1 if the attempt succeeded, else returns a 0. Getting the bits returns a -1 if it failed. Parameters:

safe - change protection even if in PRETEND mode

`(abort <message> <message> ...)` - Exits the installation procedure with the given messages and then processes the onerror statements (is any).

`(exit)` - This causes normal termination of a script. The "done with installation" message is displayed. The "onerror" statements are not executed.

`(complete <number>)` - This statement is used to inform the user how complete the installation is. The number (which must be between 0 and 100) will be printed in the title bar of the installer window with a '%' sign.

`(message <string> <string> ...)` - This statement displays a message to the user in a window, along with Proceed, Abort and optional Help buttons. Parameters:

help - optional help text

`(working <parameters>)` - Prints a message that the installer still at work. Useful if you are a doint a long operation other than file copying (which has its own status display). Parameters:

prompt - as above

Control Statements

NOTE: Strings can be used as the result of a test expression. An empty string is considered a FALSE value, all others are considered TRUE.

`(if <expression> <true-statement> <false-statement>)` - Operates as a standard "if-then" statement.

`(while <expression> <statement> ... )` - Operates as a standard "do-while" statement.

`(until <expression> <statement> ... )` - Operates as a standard "do-until" statement.

`(foreach <drawer name> <pattern> <statement>)` - For each file or directory matching the pattern located in the given drawer statement will be executed. The special variables "@each-name" and "@each-type" will contain the filename and the DOS object type, respectfully. (By DOS object type we mean the same value as found in fib_DirEntryType if one Examine'd the object.)

`((...) (...) (...))` - Execute a sequence of statements. The statements in the parentheses will be executed in order -- not needed at topmost level.

`(trap <trapflags> <statements>)` - Used for catching errors. Works much like C "longjmp", i.e. when an error occurs, control is passed to the statement after "trap". "Trapflags" determine which errors are trapped. The trap statement itself returns the error type or zero if no error occured. The current error type values are:

1 - user aborted
2 - ran out of memory
3 - error in script
4 - DOS error (see @ioerr below)
5 - bad parameter data

`(onerror <statements>)` - When a fatal error occurs that was not trapped, a set of statements can be called to clean-up after the script. These statements are logged in by using the onerror construct. Note that onerror can be used multiple times to allows context sensitive termination.

Functions

`(<string> <arguments> ...)` - The "string substitution function". Whenever a text string is the first item in a parenthesized group, the arguments will be substituted into the string using RawDoFmt. Note: This function does no argument type checking.

`(cat <string> <string> ...)` - Concatenates the strings and returns the resulting string.

`(transcript <string> <string> ...)` - Concatenates the strings, appends a newline and then prints the resulting string to the transcript file (if any).

`(tackon <path> <file>)` - Concatenates the filename to the pathname and returns resulting string.

`(fileonly <path>)` - Returns only the file part of a pathname.

`(pathonly <path>)` - returns only the non-file part of a pathname.

`(askdir <parameters>)` - Asks the user for a directory name, with a scrolling list requester. The user can either create a new directory or specify an existing one. If the user cancels, the routine will cause an abort. Parameters:

prompt, help - as above
default - default name of directory to be selected.

`(askfile <parameters>)` - Asks the user for a file name, with a scrolling list requester. Parameters:

prompt, help - as above
default - default name of file to be selected

`(askstring <parameters>)` - Prompts the user to enter a text string. Parameters:

prompt, help - as above
default - the default text string.

`(asknumber <parameters>)` - Prompts the user to enter an integer quantity. Parameters:

prompt, help - as above
range - valid input range of numbers
default - default value

`(askchoice <parameters>)` - Ask the user to select one out of N choices, using radio buttons. Parameters:

prompt, help - as above
choices - a list of choice strings, such as ''ok'' ''cancel'', etc.
default - the number of the default choice (defaults to 0)

`(askoptions <parameters>)` - Ask the user to select any number of N choices, using checkbox buttons. A bit mask is returned as a result, with the first bit indicating the state of the first choice, etc. Parameters:

prompt, help - as above
choices - a list of choice strings, such as ''ok'' ''cancel'', etc.
default - a bit mask of the buttons to be checked (defaults to -1)

`(askbool <parameters>)` - Ask the user to select yes or no. Parameters:

prompt, help - as above
default - 0 = no, 1 = yes
choices - change the positive and negative text. The defaults are ''Yes'' and ''No''. So to change the text to ''Proceed'' and ''Cancel'' you would use: (choices ''Proceed'' ''Cancel'')

`(askdisk <parameters>)` - Ask the user to insert a disk in a user friendly manner. For instance, the prompt can describe the disk by its label; e.g. ''FooBar Program Disk''. This function will not exit until the correct disk is inserted, or the user aborts.

prompt, help - as above

dest - the volume name of the disk to be inserted

newname - a name to assign to the disk for future reference.

This assignment is done even in Dry Run mode -- it is considered "safe" disk - switch to get a drive list to be shown initially.

Note: volume name must be supplied without a colon; i.e. "ENV" not "ENV:".

`(exists <filename>)` - Returns 0 if does not exists, 1 if a file, and 2 if a directory.

`(earlier <file-1> <file-2>)` - Returns TRUE is file-1 is earlier than file-2.

`(getsize <filename>)` - Returns the size of a file.

`(getdiskspace <pathname>)` - Returns the available space in bytes on the disk given by pathname. Returns a -1 if the pathname is bad or information could not be obtained from the filesystem (even though pathname was valid).

`(getsum <filename>)` - Returns the checksum of a file, for comparing versions.

`(getversion <filename>)` - If the named file has a RomTag with an ID string or a 2.x version string, this will return the version number. If filename is not provided, then the version of the OS is returned instead. Note that this function does *not* assume files ending with ".library" or ".device" reside a particular place -- the path must be included.

`(getenv <name>)` - Returns the contents of the given ENV: variable.

`(getassign <name> <opts>)` - Returns the pathname of the object 'name'. The default is for logical assignments only, but can be changed using an options string where the characters are:

'v' - only match volumes
'a' - only match logical assignments
'd' - only match devices

Therefore 'a' would be equalivant to having no options.

Note: name must be supplied without a colon; i.e. "ENV" not "ENV:".

`(select <n> <item> <item> ...)` - Returns the value of the Nth item.

`(=  <expression-1> <expression-2>)`
`(>  <expression-1> <expression-2>)`
`(>= <expression-1> <expression-2>)`
`(<  <expression-1> <expression-2>)`
`(<= <expression-1> <expression-2>)`
`(<> <expression-1> <expression-2>)` - These are the standard relational expressions.

`(+  <expression> ...)` - Returns the sum of all the arguments.

`(- <expression-1> <expression-2>)` - Returns the first argument minus the second argument

`(* <expression> ...)` - Returns the product of all the arguments

`(/ <expression-1> <expression-2>)` - Returns the first argument divided by the second argument

```
(NOT <expression>)
(AND <expression-1> <expression-2>)
(OR  <expression-1> <expression-2>)
(XOR <expression-1> <expression-2>)
```
- Standard logical functions

`(IN <expression> <bit numumer-1> ...)` - Returns 0 if none of the given bit numbers (starting at 0 for the LSB) is set in the result of expression, else returns a mask of the bits that were set.

Summary Of Parameters

`(help <string-1> <string-2> ...)` - This is used to specify the help text for each action.

`(prompt <string-1> <string-2> ...)` - This is used to provide the ''title'' of the screen which explains to the user what this step does.

`(safe)` - This tells the installer that an action not normally performed in Pretend mode should be performed.

`(choices <string-1> <string-2> ...)` - Used to display a series of checkmarks. This is used in the ''askchoice'' function to indicate what choices the user has. It can also be used in the ''copyfiles'' statement to specify that only certain files can be copied. (If not present, some other criterion will be used to determine which files to copy).

`(pattern <string>)` - Used in the ''copyfiles'' statement to specify a wildcard pattern.

`(all)` - In the ''copyfiles'' statement, specifies that all files are to be copied.

`(source <filename>)` - Specifies the file or directory to be read as part of this command.

`(dest <filename>)` - Specifies the file or directory to be modified as part of the command.

`(newname <name>)` - Used in ''copyfiles'' to specify that a file will have a new name after being copied. Used in ''askdisk'' in assign the new name to the inserted disk.

`(confirm <user-level>)` - On some statements, the user will only be informed of the action (and allowed to cancel it) if the ''confirm'' option is specified. The user level can be ''expert'' or ''average'' (''expert'' is the default).

`(infos)` - Indicates to the ''copyfiles'' statement that accompanying ''.info'' files are to be copied as well.

`(fonts)` - Indicates to the ''copyfiles'' statement that accompanying ''.font'' files are to be copied as well.

`(optional)` - Indicates to the ''copyfiles'' and ''copylib'' statement that it is not a fatal error to have a copy fail.

`(settooltype <tooltype> <value>)` - Used to modify a tooltype to a certain value. If the tooltype does not exist it will be created; if the <values> parameter is omitted, the tooltype will be deleted.

`(setdefaulttool <value>)` - Used to modify a the default tool of an icon.

`(setstack <value>)` - Used to modify a the stack size included in an icon.

`(noposition)` - Used to modify a positioning of an icon to NO_ICON_POSITION.

`(disk)` - n used with the ''rename'' statement, specifies that a disk relabel operation is really desired. When used with the ''askdir'' statement, specifies that a drive list should be shown initially (instead of a file list).

`(append <string>)` - Within a ''textfile'' statement, will append the string to the textfile.

`(include <filename>)` - Within a ''textfile'' statement, will append the listed file to the textfile.

`(default <value>)` - specifies the default value of an askchoice, askstring, or asknumber action.

`(range <min> <max>)` - specifies the range of allowable numbers for an asknum statement.

`(command <text> ...)` - specifies the text of a command to be inserted into the S:User-Startup file. (Argument strings are merged.)

Pre-Defined Variables

@icon
The pathname of the icon used to start the installer.

@default-dest
The installer's suggested location for installing an application. If you installed the application somewhere else (as the result of asking the user) then you should modify this value -- this will allow the "final" statement to work properly. Note that creating a drawer and putting the application in that drawer is considered installing the application somewhere else. Set it to "" if there really is no definite place that the "application" was installed. The log file will be copied to the drawer indicated by @default-dest unless it was set to "".

@pretend
The state of the pretend flag (1 if pretend mode).

@user-level
The user-level the script is being run at: 0 for novice, 1 for average, 2 for expert.

@error-msg
The text that would have been printed for a fatal error, but was overridden by a trap statement.

@special-msg
If a script wants to supply its own text for any fatal error at various points in the script, this variable should be set to that text. The original error text will be appended to the special-msg within parenthesis. Set this variable to "" to clear the special-msg handling.

@ioerr
The value of the last DOS error. Can be used in conjunction with the "trap" statement to learn more about what an error occured.

@each-name
@each-type
Used in a "foreach" loop (see above).

@askoptions-help
@askchoice-help
@asknumber-help
@askstring-help
@askdisk-help
@askfile-help
@askdir-help
@copylib-help
@copyfiles-help
@makedir-help
@startup-help
Default help text for various functions. These can be appended to the explanation provided for a particular action or used as is.

IV. Installer Language Quick Reference

Overview

Attempts to install in ''work:'' by default if it exists.

HELP key brings up context-sensitive help.

Esc key brings up the abort requester.

Can add assigns to s:User-Startup, and adds lines to s:Startup-Sequence (if necessary?) to make sure s:User-Startup is executed upon boot-up.

Can check versions of files/libraries.

Install can run in ''real'' (do it) or ''pretend'' (dry run) modes.

Quick language overview

Language is lisp-like (lots of parentheses (()) (-:).

Variables are typeless (a la ARexx), i.e., strings and numbers are treated interchangably.

Strings are delimited with " or '.

Certain embedded sequences are available for strings:

'\n' newline '\r' return
'\t' tab '\0' NUL
'\"' double-quote '\\' backslash

Statements go in parentheses ( ). The general format is:

(operator <operand1> <operand2> ...)

E.g., to assign the value '5' to the variable 'x', use

(set x 5)

To produce the sum of two numbers, use

(+ 5 9)

Note that there is no difference between operators and functions--the function 'set' and the arithmetic operator '+' are both used exactly the same way.

Combining statements: A statement can be used as the operand to another statement. E.g.: (set x (+ 3 5)). In this case, the statement '(+ 3 5)' is evaluated first, and the result is 8. You can think of this as having the '(+ 3 5)' part being replaced by an 8, leaving: (set v 8)

Note that the '(+ 3 5)' part actually produced a value: ''8''. This is called the ''result'' of the statement. Many statements return results, even some that might surprise you (such as ''set'' and ''if'').

Comments are preceeded with a semi-colon '';''

Hex numbers are preceeded with a $, e.g., $23 ; 35

Binary numbers are preceeded with a %, e.g., %0101 ; 5

Many statements return a value, which can be used in assignments, tests, etc.

Data can be formatted using a string literal with argument placemarkers, e.g.,

("i am %ld foot %ld inches tall." 6 3)
;produces a string with %ld's replaced with 6 and 3.

Pre-Defined Variables

@icon - pathname of install script icon
@default-desk - dir. where install wants to put things by default
@pretend - state of "pretend" (dry run mode) flag
@user-level - 0-Novice,1-Average,2-Expert
@error-msg - msg that would be displayed if error not trapped (see trap)
@special-msg - custom fatal error message
@each-name, @each-type - used in "foreach" loop

Default Help String Variables

@askoptions-help@askchoice-help@asknumber-help
@askstring-help@askdisk-help@askfile-help
@askdir-help @copylib-help@copyfiles-help
@makedir-help @startup-help

Statements

Many commands have standard parameters (some optional):
(all) ; specifies all files are to copied
(append <string>) ; add string to text file (for textfile)
(choices <string1> <string2> ...) ; radio button options
(command <string1> <string2>...) ; add to s:user-startup
(confirm <user-level>) ; confirmation
(default <value>) ; default value, choice, string, etc.
(dest <file>) ; output to <file>
(help <string1> <string2> ...) ; define current help info
(include <file>) ; insert file in textfile statement
(infos) ; copy .info files also
(newname <name>) ; specify new file or disk name
(noposition) ; make icon "floating"
(pattern <string>) ; used w/ "files" for patterns
(prompt <string1> <string2> ...) ; text to show user
(range <min> <max>) ; numeric input (asknum) range
(safe) ; force installer to perform action even if in pretend mode.
(settooltype <tooltype> <value>) ; set icon tool type
(setdefaulttool <value>) ; set icon's default tool
(setstack <value>) ; set icon's stack value
(source <file>) ; read from <file>

Note: Custom parameters are shown below in < >, and standard parameters are show as (param..) where "param" is one of help, prompt, safe, etc. See above for details on standard parameters.

(abort <string1> <string2> ...)
; abandon installation
(complete <num>)
; display percentage through install in titlebar
(copyfiles (prompt..) (help..) (source..) (dest..) (newname..)

```
  (choices..) (all) (pattern..) (files) (infos) (confirm..) (safe))
  ; copy files (and subdir's by default). files option say NO subdirectories
(copylib (prompt..) (help..) (source..) (dest..) (newname..) (infos)
  (confirm) (safe)
  ; install a library if newer version
(delete file (help..) (prompt..) (confirm..) (safe))
  ; delete file
(execute <arg> (help..) (prompt..) (confirm) (safe))
  ; execute script file
(exit)
  ; end installation
(foreach <dir> <pattern> <statements>)
  ;do for entries in directory
(if expr truestatements falsestatements)
  ; conditional
(makeassign <assign> <path> (safe)) ; note: <assign> doesn't need ':'
  ; create an assignment
(makedir <name> (prompt..) (help..) (infos) (confirm..) (safe))
  ; make a directory
(message <string1> <string2>...)
  ; display message with Proceed, Abort buttons
(onerror (<statements>))
  ; general error trap
(rename <old> <new> (help..) (prompt..) (confirm..) (safe))
  ; rename files
(rexx <arg> (help..) (prompt..) (confirm..) (safe))
  ; execute ARexx script
(run <arg> (help..) (prompt..) (confirm..) (safe))
  ; execute program
(set <varname> <expression>)
  ; assign a value to a variable
(startup (prompt..) (command..))
  ; add a command to the boot scripts (startup-sequence, user-startup)
(textfile (prompt..) (help..) (dest..) (append) (include..)
  (confirm..) (safe))
  ; create text file from other text files and strings
(tooltype (prompt..) (help..) (dest..) (settooltype..) (setstack..)
  (setdefaulttool..) (noposition) (confirm..) (safe))
  ; modify an icon
(trap <flags> <statements>)
  ; trap errors.  flags:1-abort,2-nomem,3-error,4-dos,5-badargs
(until <expr> <statements>)
  ; do-until conditional structure (test end of loop)
(while <expr> <statements>)
  ; do-while conditional structure (test top of loop)
(working)
  ; indicate to user that installer is busy doing things
```

Functions

```
(= <expr1> <expr2>) ; equality test (returns 0 or 1)
(> <expr1> <expr2>) ; greater than test (returns 0 or 1)
(>= <expr1> <expr2>) ; greater than or equal test (returns 0 or 1)
(< <expr1> <expr2>) ; less than test (returns 0 or 1)
(<= <expr1> <expr2>) ; less than or equal test
(+ <expr1> <expr2> ...) ; returns sum of expressions
(- <expr1> <expr2>) ; returns <expr1> minus <expr2>
(* <expr1> <expr2> ...) ; returns product of expressions
(/ <expr1> <expr2>) ; returns <expr1> divided by <expr2>
(AND <expr1> <expr2>) ; returns logical AND of <expr1> and <expr2>
(OR <expr1> <expr2>) ; returns logical OR of <expr1> and <expr2>
(XOR <expr1> <expr2>) ; returns logical XOR of <expr1> and <expr2>
(NOT <expr>) ; returns logical NOT of <expr>
(IN <expr> <bit-number> <bitnumber>...); returns <expr> AND bits
(<format string> <arg1> <arg2> ...) ; printf clone
(askdir (prompt..) (help..) (default..)) ; ask for directory name
(askfile (prompt..) (help..) (default..)) ; ask for file name
(askstring (prompt..) (help..) (default..)) ; ask for a string
(asknumber (prompt..) (help..) (range..) (default..)) ; ask for a number
(askchoice (prompt..) (choices..) (default..)) ; choose 1 options
(askoptions (prompt (help..) (choices..) default..)) ; choose n options
(askbool (prompt..) (help..) (default..) (choices..)) ; 0=no, 1=yes
(askdisk (prompt..) (help..) (dest..) (newname..))
(cat <string1> <string2>...) ; returns concatenation of strings
(exists <filename>) ; 0 if no, 1 if file, 2 if dir
(earlier <file1> <file2>) ; true if file1 earlier than file2
(fileonly <path>) ; return file part of path (see pathonly)
(getassign <name> <opts>) ; return value of logical name (no ':')
 ; <opts: 'v'-volumes, 'a'-logical, 'd'-devices
(getdiskspace <path>) ; return available space
(getenv <name>) ; return value of environment variable
(getsize <file>) ; return size
(getsum <file>) ; return checksum of file for comparison purposes
(getversion) ; return version of file, library, etc.
(pathonly <path>) ; return dir part of path (see fileonly)
(select <n> <item1> <item2> ...) ; return n'th item
(transcript <string1> <string2>) ; puts concatenated strings in log file.
(tackon <path> <file>) ; return properly concatenated file to path
```

IFF--Past, Present, and Future

by Chris Ludwig

Written by Electronic Arts in 1985, the original Interchange File Format document (EA85 IFF) proposed a way of allowing users to exchange data between various programs. A mechanism was also provided for the standardized addition of new data formats.

Today, IFF is an integral part of the Amiga's success. Thanks to the continues support for IFF in the developer community, no other computing platform offers the promise of so much connectivity between applications.

The "New" IFF Sample Code

Supporting some of the more subtle aspects of the IFF system has not always been particularly easy. Commodore's addition of the *iffparse.library* to the operating system should make it easier for developers to deal with things like LISTs, CATs, and nested FORMs.

From the beginning, developers have relied on the sample code provided in the original IFF specification as their starting point for developing IFF support code. Unfortunately, this original code may not have been particularly clear to all developers. New IFF sample code has been written with the dual goals of furthering the use of *iffparse.library*, and providing developers with new, more readable (and more complete) examples of IFF and clipboard usage.

The sample code also takes advantage of a number of new system software features which are available only under realease 2.0 (V36) or greater of the OS. One important feature is that the new sample code which deals with ILBM FORMs exploits the built-in *Display Database* of release 2.0. This allows the code to intelligently display ILBM pictures on any Amiga computer, and will probably allow the code to have a high degree of compatibility with any future Amiga graphics hardware that might be developed.

Creating New FORMS

When developing a new IFF type, there are several steps you should follow:

- Discuss needs and specifications within the developer community and with Commodore.

  The most important thing about designing IFF FORMs and chunks is that they meet the data storage and transfer needs of multiple applications. When more than one product uses the same IFF type, the market widens for all products that use that IFF type. Users are not forced to use one product or another, but can buy as many as they need to get the job done, fully utilizing all the features that each product has to offer. This step helps to ensure that a proposed IFF form or chunk type is flexible and isn't redundant. A good way to start this kind of discussion is to post a message in Commodore's amiga.dev/iff conference on the BIX electronic network. Also, feel free to contact me to discuss your IFF needs and issues. You can reach me at (215) 431-9316 or on BIX (cludwig). I can also be reached via UseNet InterNet e-mail at:

  chrisl@cbmvax.commodore.com or ...[uunet| rutgers]!cbmvax!chrisl.

- Implement the new type and conduct feasibility tests.

  Before settling on a format, set up prototype code to test the proposed format. This will help to prove that the idea is sound and can be implemented in software before others try to use it.

- Submit specifications to Commodore.

  Coming up with a new kind of IFF FORM or chunk is easy--almost too easy. Just about anyone can follow the IFF guidelines and define their own FORM or chunk. If every application used a different IFF FORM, one application would be unable to share data with another because it can't read the other application's IFF FORM. It's like making up a new word for something that everyone sees every day. You may understand what the word means, but when you try to use your new word to communicate with others, they won't understand you. Further, deciding to use a pre-existing FORM or chunk in a new and different way is a lot like making up your own meaning for a pre-existing word. Confusion results when programs try to read FORMs or chunks whose meaning was altered by a non-conforming program.

  To avoid the problem of incompatible IFF types, register your new IFF types with CATS. CATS acts as a "dictionary" of IFF types. By submitting your proposals for FORM or chunk types to CATS, you help prevent duplication of an existing data type. Also, if you register your new IFF type, it is more likely that it will be adopted as an IFF standard that other applications will use. For example, the ANIM form came from third party developers who proposed and refined the format. Now ANIM is the de facto standard for animation files.

  CATS wants this last step to be as easy as possible, so we've included a standard form at the end of this article. Just photocopy the form whenever you need it, and use it as a guide for submitting your FORM or chunk specifications. The registration form should be accompanied by a disk containing ASCII text files of the IFF specification. If available, include some code examples which demonstrate the use of the IFF type. Please do not place copyright notices in your specifications or examples so that CATS may make them available to other developers.

For an excellent example of a third party FORM specification, see the WORD FORM in the third party specification section of the IFF manual. For an example of chunk descriptions, examine the 8SVX FORM's SEQN and FADE chunks in the same section.

Note that even if you don't plan to release the specifications of your FORM or chunk, you must still register the name with CATS. This is the only way to prevent name conflicts in IFF files. You should register your FORM and chunk names before finalizing your product and its documentation in case there is a name conflict with an existing IFF type.

- Distribute final specifications to the developer community.

  Once you have registered your FORM or chunk with CATS, you should release the specifications of the chunk to the developer community. Although CATS publishes FORMs and chunks in the *IFF Manual* and occasionally in *Amiga Mail*, developers should not rely on these methods to distribute their IFF type specification. One of the most efficient ways to distribute your specification is to include it in your application's documentation. Another good distribution method is to post the final specifications in amiga.dev/iff on BIX. Distributing the specification will increase the probability of your form or chunk becoming a standard.

What's Needed (Suggestions for new IFF FORMs and chunks)

Nearly every developer has an idea about ''what the Amiga needs'', so let's take a look at the areas where there's at least a percieved need, and try to come up with some solutions.

A. JPEG (JFIF)
B. Animation on the Workbench
C. Time sensitive Video with audio
    1. Time stamping of frames
D. More support for clipboard
    1. 8SVX
    2. FTXT
    3. ILBM
    4. DR2D
    5. 3D standard (TDDD? 3DG? Interchange?)

Conclusion

With more users demanding connectivity between their various applications, how well YOUR application supports IFF and the clipboard could well set it apart from others.

CATS IFF FORM/Chunk Registration Form

How to register an IFF FORM or Chunk:

- Make Sure your FORM/Chunk names follow the naming conventions in the document *EA IFF 85 Standard for Interchange Format Files* (available from CATS) and also in the 1.3 *RKM: Includes & Autodocs* IFF Appendix.
- Call CATS at (215) 431-9300 to find out if you FORM/Chunk name conflicts with any existing name.
- Fill out this form.
- Mail this form and and an AmigaDOS disk containing the ASCII text specification of your FORMs/Chunks (plus any include files and sample source code) to:

CATS - IFF Registration
Commodore Business Machines, Inc.
1200 Wilson Drive
West Chester, PA 19380
USA

Date ____/____/____ Company Name ____________________

Contact ____________________

Address ____________________

City, State, ZIP ____________________

Telephone ____________ CATS Developer Number ____________

☐ New Form Name ________ Brief Description ________________

☐ New Chunk(s) Name ________ Brief Description ________________

for FORM: ________ Name ________ Brief Description ________________

Name ________ Brief Description ________________

Name ________ Brief Description ________________

Software Titles that support this FORM/Chunk: ____________________

____

Check the following that apply:

☐ Form/Chunk specification enclosed on disk ☐ Code examples enclosed on disk

☐ FORM/Chunk specification is proprietary and should not be released

Appendix A - DRNG Chunk for FORM ILBM

Submitted by Lee Taran

Purpose: Enhanced Color Cycling Capabilities

DPaintIV supports a new color cycling model which does *not* require that color cycles contain a contiguous range of color registers. For example, If your range looks like:

```
[1][3][8][2]
```

then at each cycle tick

```
 temp = [2],
 [2] = [8],
 [8] = [3],
 [3] = [1],
 [1] = temp
```

You can now cycle a single register thru a series of RGB values. For example, if your range looks like:

```
[1] [orange] [blue] [purple]
```

then at each cycle tick color register 1 will take on the next color in the cycle:

```
ie:  t=0:  [1] = curpal[1]
t=1:  [1] = purple
t=2:  [1] = blue
t=3:  [1] = orange
t=4:  goto t=0
```

You can combine rgb cycling with traditional color cycling. For example your range can look like:

```
[1] [orange] [blue] [2] [green] [yellow]

t=0: [1] = curpal[1], [2] = curpal[2]
t=1: [1] = yellow,    [2] = blue
t=2: [1] = green,     [2] = orange
t=3: [1] = curpal[2], [2] = curpal[1]
t=4: [1] = blue,      [2] = yellow
t=5: [1] = orange,    [2] = green
t=6: goto t=0
```

Note: DPaint will save out an old style range CRNG if the range fits the CRNG model otherwise it will save out a DRNG chunk. No thought has been given (yet) to interlocking cycles.

```
/* ---------------------------------------------------------------------

 IFF Information:  DPaintIV DRNG chunk

          DRNG ::= "DRNG" # { DRange DColor* DIndex* }

 a <cell> is where the color or register appears within the range

 The RNG_ACTIVE flags is set when the range is cyclable. A range
 should only have the RNG_ACTIVE if it:
      1> contains at least one color register
      2> has a defined rate
      3> has more than one color and/or color register
 If the above conditions are met then RNG_ACTIVE is a user/program
 preference.  If the bit is NOT set the program should NOT cycle the
 range.

 The RNG_DP_RESERVED flags should always be 0!!!
 -------------------------------------------------------------------- */
typedef struct {
   UBYTE min;           /* min cell value */
   UBYTE max;           /* max cell value */
   SHORT rate;          /* color cycling rate, 16384 = 60 steps/second */
   SHORT flags;         /* 1=RNG_ACTIVE,4=RNG_DP_RESERVED */
   UBYTE ntrue;         /* number of DColor structs to follow */
   UBYTE nregs;         /* number of DIndex structs to follow */
   } DRange;

typedef struct { UBYTE cell; UBYTE r,g,b; } DColor; /* true color cell */
typedef struct { UBYTE cell; UBYTE index; } DIndex; /* color register cell */
```

Appendix B - IFF FORM and Chunk Registry - August 1, 1991

The following is an alphabetical list of registered FORMs, generic chunks (shown as (any).chunkname), and registered new chunks for existing FORMs (shown as formname.chunkname). The center column describes where additional information on the FORM or chunk may be found. Items marked "EA_IFF" are described in the main chapters of the EA IFF specs. Those marked "TP_SPECS" are be described in the third-party specications section. Items marked "propos" are proposals which have been submitted to CATS, some of which are private. And items marked with "------" are private or yet unreleased specifications. New chunks and FORMS should be registered with CATS US, IFF Registry, 1200 Wilson Drive, West Chester, PA. 19380. Please make all submissions on Amiga diskette and include your address, phone, fax.

(any).ANNO	EA_IFF	EA IFF 85 Generic Annotation chunk
(any).AUTH	EA_IFF	EA IFF 85 Generic Author chunk
(any).CHRS	EA_IFF	EA IFF 85 Generic character string chunk
(any).CSET.doc	IFF_TP	chunk for specifying character set
(any).FVER.doc	IFF_TP	chunk for 2.0 VERSION string of an IFF file
(any).NAME	EA_IFF	EA IFF 85 Generic Name of art, music, etc. chunk
(any).TEXT	EA_IFF	EA IFF 85 Generic unformatted ASCII text chunk
(any).(c)	EA_IFF	EA IFF 85 Generic Copyright text chunk
8SVX	EA_IFF	EA IFF 85 8-bit sound sample form
8SVX.CHAN.PAN.doc	IFF_TP	Stereo chunks for 8SVX form
8SVX.SEQN.FADE.doc	IFF_TP	Looping chunks for 8SVX form
ACBM.doc	IFF_TP	Amiga Contiguous Bitmap form
AHAM	----	unregistered (???)
AIFF.doc	IFF_TP	Audio 1-32 bit samples (Mac,AppleII,Synthia Pro)
ANBM.doc	IFF_TP	Animated bitmap form (Framer, Deluxe Video)
ANIM.brush.doc	IFF_TP	ANIM brush format
ANIM.doc	IFF_TP	Cel animation form
ANIM.op6	----	ANIM opt 6 reserved for stereo opt 5 ANIMs
ARC.proposal	propos	archive format proposal(old)
ARES	----	unregistered (???)
ATXT	----	temporariliy reserved
AVCF	----	AmigaVision flow (format not yet released)
AVCO	----	AmigaVision commands (format not yet released)
AVEV	----	AmigaVision events (format not yet released)
BANK	----	Soundquest Editor/Librarian MIDI Sysex dump
BBSD	----	BBS Database, F.Patnaude,Jr., Phalanx Software
C100	----	Cloanto Italia private format
CAT	EA_IFF	EA IFF 85 group identifier
CHBM	----	Chunky bitmap (name reserved by Eric Lavitsky)
CLIP	----	CAT CLIP to hold various formats in clipboard
CPFM	----	Cloanto Personal FontMaker (doc in their manual)
DCCL	----	DCCL - DCTV paint clip
DCPA	----	DCPA - DCTV paint palette
DCTV	----	DCTV - DCTV raw picture file
DECK	----	private format for Inovatronics CanDo
DR2D.doc	IFF_TP	2-D Object standard format
DRAW	----	reserved by Jim Bayless, 12/90
FANT.doc	IFF_TP	Fantavision movie format
FIGR	----	Deluxe Video - reserved
FNTR	EA_IFF	EA IFF 85 reserved for raster font
FNTV	EA_IFF	EA IFF 85 reserved for vector font
FORM	EA_IFF	EA IFF 85 group identifier
FTXT	EA_IFF	EA IFF 85 formatted text form
GRYP.proposal	propos	byteplane storage proposal (copyrighted)
GSCR	EA_IFF	EA IFF 85 reserved gen. music score
GUI.proposal	propos	user interface storage proposal (private)
HEAD.doc	IFF_TP	Flow - New Horizons Software
ILBM	EA_IFF	EA IFF 85 raster bitmap form
ILBM.3DCM	----	reserved by Haitex
ILBM.3DPA	----	reserved by Haitex
ILBM.ASDG	----	private ASDG application chunk
ILBM.BHBA	----	private Photon Paint chunk (brushes)
ILBM.BHCP	----	private Photon Paint chunk (screens)
ILBM.BHSM	----	private Photon Paint chunk
ILBM.CLUT.doc	IFF_TP	Color Lookup Table chunk
ILBM.CMYK.CNAM.pro	propos	CMYK Cy/Mag/Yel cmap, CNAM color names
ILBM.CTBL.DYCP.doc	IFF_TP	Newtek Dynamic Ham color chunks
ILBM.DCTV	----	reserved
ILBM.DGVW	----	private Newtek DigiView chunk
ILBM.DPI.doc	IFF_TP	Dots per inch chunk
ILBM.DPPV.doc	IFF_TP	DPaint perspective chunk (EA)
ILBM.DRNG.doc	IFF_TP	DPaint IV enhanced color cycle chunk (EA)
ILBM.EPSF.doc	IFF_TP	Encapsulated Postscript chunk
ILBM.TMAP	----	Transparency map (temporarily reserved)

ILBM.VTAG.proposal	propos	Viewmode tags chunk suggestion
IOBJ	----	reserved by Seven Seas Software
ITRF	----	reserved
LIST	EA_IFF	EA IFF 85 group identifier
MIDI	----	Circum Design
MOVI	----	LIST MOVI - private format
MSCX	----	private Music-X format
MSMP	----	temporarily reserved
MTRX.doc	IFF_TP	Numerical data storage (MathVision - Seven Seas)
NSEQ	----	Numerical sequence (Stockhausen GmbH)
OCMP	EA_IFF	EA IFF 85 reserved computer prop
OCPU	EA_IFF	EA IFF 85 reserved processor prop
OPGM	EA_IFF	EA IFF 85 reserved program prop
OSN	EA_IFF	EA IFF 85 reserved serial num prop
PGTB.doc	IFF_TP	Program traceback (SAS Institute)
PICS	EA_IFF	EA IFF 85 reserved Macintosh picture
PLBM	EA_IFF	EA IFF 85 reserved obsolete name
PROP	EA_IFF	EA IFF 85 group identifier
PRSP.doc	IFF_TP	DPaint IV perspective move form (EA)
PTCH	----	Patch file format (SAS Institute)
PTXT	----	temporarily reserved
README	----	
RGB4	----	4-bit RGB (format not available)
RGBN-RGB8.doc	IFF_TP	RGB image forms, Turbo Silver (Impulse)
RGBX	----	temporarily reserved
ROXN	----	private animation form
SAMP.doc	IFF_TP	Sampled sound format
SC3D	----	private scene format (Sculpt-3D)
SHAK	----	private Shakespeare format
SMUS	EA_IFF	EA IFF 85 simple music score form
SYTH	----	SoundQuest Master Librarian MIDI System driver
TDDD.doc	IFF_TP	3-D rendering data, Turbo Silver (Impulse)
UNAM	EA_IFF	EA IFF 85 reserved user name prop
USCR	EA_IFF	EA IFF 85 reserved Uhuru score
UVOX	EA_IFF	EA IFF 85 reserved Uhuru Mac voice
VDEO	----	private Deluxe Video format
WORD.doc	IFF_TP	ProWrite document format (New Horizons)

The New IFFParse Support Modules

by Carolyn Scheppner

The new *iffparse.library* code modules and examples are designed as replacements for the original EA IFF code. For some modules, it has been possible to retain much of the original code. However, most structures and most higher level function interfaces have changed.

On the plus side, these new modules contain many new high-level, easy-to-use functions for querying, loading, displaying, and saving ILBMs. During their development, modules similar to these have been used inhouse at Commodore for the 2.0 *Display* program and several other ILBM applications. The *screen.c* module provides powerful display-opening functions which are 1.3-compatible yet provide a host of new options under 2.0 such as centered overscan screens, full video display clips, border transparency control, and autoscroll. New modules have been added for printing (screendump) and for preserving/adding chunks (copychunks). And the 8SVX example now actually plays samples and instruments. In addition, clipboard support is automatic for all applications that use the IFFP modules because *parse.c's* openifile() interprets the filename -c[n] (ie. "-c", "-c1", "-c2", etc.) as clipboard unit n.

Most of the modules and examples require *iffparse.library* which is distributed with Workbench 2.0. Please note that *iffparse.library* is a 1.3-compatible library, and that all of the modules and examples have been designed to take advantage of 2.0, but also work under 1.3. Developers who wish to distribute *iffparse.library* on their commercial products may execute a 2.0 Workbench license, or may get an addendum to their 1.3 Workbench license to allow distribution of *iffparse.library*.

It was not necessary to port the gio IO speedup code since iffparse can use your compiler's own buffered IO routines through the callback stdio_stream() in *parse.c*. Depending on your application, you may want to add your own additional buffering to this stdio_stream() code.

Most of the high-level function pairs provided in these modules have been designed to provide safe cleanup for themselves. For example, a loadbrush() that succeeds or fails at any point can be cleaned up via unloadbrush. The cleanup routines null out the appropriate pointers so that allocations will not be freed twice.

All applications are built upon the *parse.c* module. The basic concept of using the *parse.c* module are:

- Define tag-like arrays of your desired chunks (readers only)
- Allocate one or more [form]Info structures as defined in *iffp/[form]app.h* (for example an ILBMInfo defined in *iffp/ilbmapp.h*).
- Initialize the ParseInfo part of these structures to the desired chunk arrays, and to an IFFHandle allocated via iffparse AllocIFF().
- Use the provided high level load/save functions, or use the lower level *parse.c* openifile(), parseifile()/getcontext()/ nextcontext() (readers only), and closeifile(). The filename c[n] may be used to read/write clipboard unit n.
- Clean up, FreeIFF(), and deallocate [form]Info's.

IMPORTANT NOTES - Most of the higher-level load functions keep the IFFHandle (file or clipboard) open. While the handle is open, you may use *parse.c* functions (such as findpropdata) *or* direct iffparse functions (FindProp(), FindCollection()) for accessing the gathered chunks. However, it is not a good idea to keep a filehandle or the clipboard open. While a clipboard unit is open, no other applications can clip to the unit. And while a file is open, you can't write the file back out. So, instead of keeping the file or unit open, you can use copychunks (in *copychunks.c*) to create a copy of your gathered chunks, and do an early closeifile() (*parse.c*). Then access, write back out (if you wish), and deallocate your copied chunks via the routines in the copychunks module (findchunk, writechunklist, freechunklist).

Warning Regarding Complex Forms

The *parse.c* module will enter complex formats such as CATSs, LISTs, and nested FORMs to find the FORM that you are interested in. This is great. However, if you are a read-modify-write program, you should warn your user when this occurs unless *you* are capable of recreating the complex format. Otherwise, your user may unknowingly destroy his complex file by writing over it with your program. For example, a paint program could read an ILBM out of a complex LIST containing pictures and music, and then save it back out as a simple ILBM, causing the user to lose his music and other pictures. To determine if a complex form was entered after a load, check the (form)Info.ParseInfo.hunt field. If TRUE (non-zero), then your file was found inside a complex format.

List of IFFP Modules and Applications

NOTE - Some useful functions are listed with each module. See module source code for docs on each function.

Applications (these require linkage with modules - see Makefiles)

ILBMDemo	Displays an ILBM, loads a brush, saves an ILBM, opt. print
ILBMLoad	Queries an ILBM and loads it into an existing screen
ILBMtoC	Outputs an ILBM as C source code
ILBMtoRaw	Converts an ILBM to raw plane/color file
RawtoILBM	Converts raw plane/color file (from ILBMtoRaw) to an ILBM
24bitDemo	Saves a simple 24-bit ILBM and then shows it 4 planes at a time (if given filename, just shows the file)
Play8SVX	Reads and plays an 8SVX sound effect or instrument. LoadSample, UnloadSample, PlaySample, OpenAudio, CloseAudio, and body load/unpack functions
ScreenSave	Save the front screen as an ILBM, with an icon

Other Examples (use *iffparse.library* directly and require no modules)

Sift	Checks and prints outline of any IFF file (uses RAWSTEP)
ILBMScan	Prints out useful info about any ILBM
ClipFTXT	Demonstrates simply clipping of FTXT to/from clipboard
cycvb.c	Dan Silva's routine for interrupt based color cycling
apack.asm	Dr. Gerald Hull's assembler replacement for *packer.c*

General Iffparse Support Module

parse.c	File/clipboard IO and general parsing openifile, closeifile, parseifile, getcontext, nextcontext, contextis, currentchunkis, PutCk chunk writing function, and IFFerr text error routine

ILBM Read Modules

loadilbm.c	High level ILBM load routines which are passed filenames (calls getbitmap) loadbrush/unloadbrush, loadilbm/unloadilbm, and queryilbm
getbitmap.c	brush/bitmap loading (non-display, calls ilbmr.c) createbrush/deletebrush, getbitmap/freebitmap
getdisplay.c	bitmap load/display (calls screen.c, ilbmr.c) showilbm/unshowilbm, createdisplay/deletedisplay

screen.c	1.3/2.0 ECS/non-ECS compatible screen/window module opendisplay, openidscreen, modefallback, clipit
ilbmr.c	Lower level ILBM body/color load routines (calls *unpacker.c*) loadbody, loadcmap, getcolors/freecolors, alloccolortable, getcamg (gets or creates modeid)
unpacker.c	BODY unpacker

ILBM Write Modules

saveilbm.c	High level ILBM saving routines which are passed filenames (calls *ilbmw.c*) screensave and saveilbm
ilbmw.c	Lower level ILBM body/color save routines (calls *packer.c*) InitBMHD, PutCMAP, PutBODY
packer.c	BODY packer

Extra Modules

copychunks.c	Chunk cloning and chunk list writing routines copychunks, findchunk, writechunklist, freechunklist
screendump.c	Screen printing module (iffparse not required)
bmprintc.c	Module to output ILBM as C code

Include Files

iffp/#?.h	This subdirectory may be kept in your current directory or in your main include directory.

Thanks to Steve Walton for his code changes for Manx/SAS compatibility, and to Bill Barton and John Bittner for their comments and suggestions.

◆

2.0 Compatibility Problem Areas

by Carolyn Scheppner, Bryce Nesbitt, Peter Cherna, and Darren Greenwald

General Compatibility Problem Areas

One sure fire way to write incompatible software is to fail to follow the Amiga programming guidelines listed in the beginning of your *Amiga ROM Kernel and Amiga Hardware* manuals. Please read the guidelines and follow them!

The following improper Amiga programming practices are likely or certain to fail on new ROMs or hardware.

☐ Calling ROM code directly.

☐ Directly or indirectly reading or writing random memory addresses or low memory (especially location zero) due to improperly initialized pointers or structures. Use Mungwall and Enforcer!!!!

☐ Assuming addresses/location/amounts of RAM or system structures.

☐ Requiring all free RAM.

☐ Mishandling 32-bit addresses. For example, using signed math or signed comparisons on addresses, or doing a BOOL or WORD test to determine if a pointer is non-zero.

☐ Overwriting memory allocations. With 32-bit addresses, a 1-byte overwrite of a string array can wipe out the high byte of a pointer or stack return address. This bug could go unnoticed on a 24-bit address machine (eg. A500,A2500, etc) but crash the system or cause other problems on an A3000.

☐ Shaving stack size too close. System function stack usage changes with each OS release.

☐ Improper flags or garbage in system structures. A bit that means nothing under one OS may drastically change the behavior of a function in a newer version of the OS. Clear structures before using, and use correct flags.

☐ Garbage passed in significant but previously unused upper bytes of function arguments (for example - the upper word of the ULONG AvailFonts() Flags parameter).

☐ Improper register or condition code handling. Do not assume registers D0-D1/A0-A1 are preserved after system calls! Some function calls happen to preserve some registers... this may change in revisions of the OS or because of system activity. In some cases we have modifed register returns to keep broken applications running under 2.0. We do not guarantee those modifications will remain in place. Condition codes are also in an undefined state on the return from a system call. Assembler code must test (TST,MOVE,CMP,etc.) D0 results before branching on condition codes. Use Scratch by Bill Hawes (via the scratchall script) to catch scratch register misuse in assembler code.

☐ Misuse of function return values. Use function prototypes and read the Autodocs for the functions you are using. Some system functions return just success or failure, or nothing at all (void). In such cases, the value which the function happens to return must not be used except as it is documented.

☐ Assembler calling of system library functions without placing the library base pointer in A6. All system functions may assume that their library's base pointer is in A6. A function's need to reference its library base can change in different OS revisions.

☐ Depending on unsupported side effects or undocumented behavior. Be sure to read the RKM chapters, Autodocs, and include file comments.

☐ Poking/peeking private system structures. These structures and the system usage of variables and flags in these structures change. Do not poke or peek any system structure unless told to do so in official Commodore documentation.

☐ Assuming current choices, configurations or initial values. If the current possibilities are A, B, or C, do not assume C if it isn't A or B. Check specifically for the choices currently implemented, and provide default behavior for unexpected values.

☐ Failure to properly allocate resources before using them.

☐ Failure to properly close/deallocate resources.

☐ Improper reading/writing of hardware registers. You must mask out bits you are not interested in on reads, and write undefined bits as zero.

☐ Assuming initial values of hardware registers. If you are going direct to the hardware, do not depend on the initial values of any hardware registers. The settings may not be the same on different versions of the OS or from bot to boot. Always set up all of the hardware registers that affect your code.

☐ Processor speed dependencies such as software delay loops.

☐ Processor instruction dependencies. Do not use instructions which are privileged on any Motorola 68xxx family processor. Do not use CLR on a hardware register which is triggered by any access (use MOVE #0 instead). The 68000 CLR instruction performs two accesses (Read, then Write). The 68020 and higher CLR instruction performs just one access.

☐ Depending on or failing to account for cache or prefetch effects. Self-modifying or trackdisk-loaded code requires cache flushes (see the exec.library/CacheClearU() function).

Amiga debugging tools such as Enforcer, Mungwall, and Scratch can find many program bugs that may affect compatibility. A program that is Enforcer/Mungwall/Scratch clean stands a much better chance of working well under current and future versions of the OS.

2.0 Changes That Can Affect Compatibility

There are several 2.0-specific areas where OS changes and enhancements can cause compatibility problems for some software.

Exec

☐ *Do not* jump to location $FC0002 as part of performing a system RESET. Many RESET functions jumped to what was the start of the ROM under 1.3. The 2.0 ROM is twice the size. We've added a *temporary* compatibility hack called "Kickety-Split" to the 2.04 Kickstart ROM. The ROM is split into two halves with a redirecting jump at $FC0002. This hack does *not* appear on the A3000 and due to space considerations will *not* appear on future machines.

☐ Everything has moved.

☐ ExecBase is moved to expansion memory if possible. Previously, ExecBase would only end up in one of two fixed locations. As a result, ColdCapture may be called after expansion memory has been configured. Great pains were taken to make this compatible.

☐ Exception/Interrupt vectors may move. This means the 68010 and above Vector Base Register (VBR) may contain a non-zero value. Poking assumed low memory vector addresses may have no effect. You must read the VBR on 68010 and above to find the base.

☐ No longer tolerant of wild Forbid() counts. Under 1.3, sometimes this bug could go unnoticed. Make sure that all Forbid's are matched with one and only one Permit (and vice versa).

☐ When an exec device gets an IORequest, it must validate io_Command. If the io_Command is 0 or out of range, the device must return IOERR_NOCMD and take no other action. The filesystem now sends new commands and expects older devices to properly ignore them.

☐ A 2.0 fix to task-switching allows a busy task to properly regain the processor after an interrupt until either its quantum (4 vblanks) is up or a higher priority task preempts it. This can dramatically change the behavior of multitask programs where one task busyloops while another same-priority task Waits. See Task Switching in the Additional Information section at the end of this document.

Expansion

☐ ExpansionBase is private - Use FindConfigDev()

☐ Memory from contiguous cards is automatically merged into one memory pool.

Strap

☐ romboot.library is gone.

☐ audio.device cannot be OpenDevice'd by a boot block program because it is not yet InitResident'd. If OpenDevice() of audio.device fails during strap, you must FindResident/InitResident audio.device, and then try OpenDevice again.

☐ boot from other floppies (+5,-10,-20,-30) is possible.

☐ undocumented system stack and register usage at Diag and Boot time have changed.

DOS

☐ DOS is now written in C and assembler, not BCPL. The BCPL compiler artifact which caused D0 function results to also be in D1 is gone. 2.0 compatibility patches which return some DOS function results in both D0 and D1 are not guaranteed to remain in the next release. Fix your programs! Use Scratch to find these problems in your code.

☐ Now has real library base with normal LVO vectors.

☐ Stack usage has all changed (variables, direction).

☐ New packet and lock types. Make sure you are not passing stack garbage for the second argument to Lock().

☐ Process structure is bigger, roll-your-own from a Task fails.

☐ Unless documented otherwise, you must be a process to call DOS functions. DOS function dependence on special process structures can change with OS revisions.

Audio.device

☐ Now not initialized until used.

☐ This means low memory open failure is possible. Check your return values from OpenDevice().

☐ This also means audio.device cannot be opened during 2.0 Strap unless InitResident'd first. If OpenDevice() of audio.device fails during strap, you must FindResident/InitResident audio.device, and then try OpenDevice again.

☐ There will be a small memory loss (until reboot) generated by the first opener of audio or narrator device (memory used in building of audio.device's base).

Gameport.device

☐ Initial state of hardware lines may differ.

Serial.device

☐ Clears io_Device on CloseDevice() (since 1.3.2)

Timer.device

❑ The most common mistake programmers make with timer.device is to send off a particular timerequest before the previous use of that timerequest has completed. Use IO_Torture to catch this problem.

❑ IO_QUICK requests may be deferred and be replied as documented.

❑ VBLANK timer requests, as documented, now wait at least as long as the full number of VBlanks you asked for. Previously, a partial vertical blank could count towards your requested number. The new behavior is more correct and matches the docs. But it can cause VBlank requests to now take up to 1 VBlank longer under 2.0. as compared to 1.3. For example, a 1/10 second request, may take 6-7 Vblanks instead of 5-6 VBlanks, or about 15% longer.

Trackdisk

❑ Private trackdisk structures have changed. See trackdisk.doc for a compatible REMCHANGEINT.

❑ Buffer is freeable, so lo-mem open failure is possible.

❑ *Do not* disable interrupts (any of them) then expect trackdisk to function while they are disabled.

Cia Timers

❑ System use of CIA timers has changed. Don't peek timers you think the system is using in a particular manner.

❑ Don't depend on initial values of CIA registers.

❑ Don't mess with CIABase. Use cia.resource.

❑ If your code requires a hardware level CIA timers, allocate the timers using cia.resource AddICRVector()! Very important! Operating system usage of the CIA timers has changed. The new 2.0 timer.device ("Jumpy the Magic Timer Device") will try to jump to different CIA's so programs that properly allocate timers will have a better chance of getting what they want. If possible, be flexible and design your code to work with whatever timer you can successfully allocate.

Other Hardware Issues

☐ Battery-backed clock is different on A3000. Use battclock.resource to access.

☐ A 68030 hardware characteristic causes longword-aligned longword writes to allocate a valid entry in the data cache, _even_ _if_ the hardware area shouldn't be cached. This can cause problems for IO registers and shared memory devices. To solve this: Don't do that OR Flush the cache OR Use Enforcer Quiet See the Motorola 68030 manual under the description of the Write Allocate bit (which must be set for the Amiga to run with the Data Cache).

Intuition

☐ Private IBase variables have moved/changed. Reading them is illegal. Writing them is both illegal and dangerous.

☐ Poking IBase MaxMouse variables is now a no-op, but please stop poking when Intuition version is >35.

☐ If you are opening on the Workbench screen, be prepared to handle larger screens, new modes, new fonts, and overscan. Also see Font compatibility information.

☐ Screen TopEdge and LeftEdge may be negative.

☐ LeftAmiga-Select is used for dragging large screens. Do not use LeftAmiga-key combinations for application command keys. The LeftAmiga key is reserved for system use.

☐ For compatibility, GetScreenData() lies if Workbench screen is a new mode. It will try to return the most sensible mode that old OpenScreen() can open. This was required to keep Workbench cloners out of difficulty. To properly handle new modes, see LockPubScreen() and GetVPModeID(), and the SA_DisplayID tag for OpenScreenTags().

☐ Using combined RAWKEY and VANILLAKEY now gives VANILLAKEY messages for regular keys, and RAWKEY messages for special keys (fkeys, help, etc.)

☐ Moving a SIMPLE_REFRESH window does not necessarily cause a REFRESHWINDOW event because layers now preserves all the bits it can.

☐ Sizing a SIMPLE_REFRESH window will not clear it.

☐ MENUVERIFY/REQVERIFY/SIZEVERIFY can time out if you take too long to ReplyMsg().

☐ Menu-key equivalents are ignored while string gadgets are active.

☐ You can't type control characters into string gadgets by default. Use CTRL-Amiga-char to type them in or use IControl Prefs to change the default behavior.

☐ Width and Height parameters of AutoRequest are ignored.

☐ New default colors, new gadget images.

☐ JAM1 rendering/text may be invisible over default colors.

☐ The cursor for string gadgets can no longer reside outside the cleared container area. If your gadget is (for example) 32 pixels wide, with MaxChars of 4, then all 32 pixels will be cleared, instead of just 24, as in 1.3.

☐ Applications and requesters that fail to specify desired fonts will get user 2.0 Font Pref fonts that may be much larger or proportional in some cases. Screen and window titlebars (and their gadgets) will be taller when accomodating a larger font. Applications which open on the Workbench screen must adapt to variable size titlebars. Any application which accepts system defaults for its screen, window, menu, Text or IntuiText fonts must adapt to different fonts and titlebar sizes. String gadgets whose height is too small for a font will revert to a smaller ROM font. There are now 2 different user-specifiable default system fonts which affect different Intuition features. This can lead to mismatches in mixed gadgets and text. For more information on where various system fonts come from and how they can be controlled, see Intuition Fonts in the Additional Information section at the end of this document.

☐ Don't modify gadgets directly without first removing them from the gadget list, unless you are using a system function designed for that purpose, such as NewModifyProp or SetGadgetAttrs.

☐ Don't rely on NewModifyProp to fully refresh your prop gadget after you've changed values in the structure. NewModifyProp will only correctly refresh changes which were passed to it as parameters. Use Remove/Add/RefreshGList for other kinds of changes.

☐ Custom screens must be of type CUSTOMSCREEN or PUBLICSCREEN. Other types are illegal. One application opens their screen with NewScreen.Type = 0 (instead of CUSTOMSCREEN, 0x0F). Then, when they open their windows, they specify NewWindow.Type of 0 and NewWindow.Screen of NULL, instead of Type=CUSTOMSCREEN and Screen=(their screen). That happened to work before, but they are now broken.

☐ Referencing IntuiMessage->IAddress as a Gadget pointer on non-Gadget IDCMP messages, or as a Window pointer (rather than looking at the proper field IntuiMessage->IDCMPWindow) may now cause Enforcer hits or crashes. The IAddress field used to always contain a pointer of some type even for IDCMP events for which no IAddress value is documented. Now, for some IDCMP events IAddress may contain other data (a non-address, possibly odd value).

☐ Using Intuition flags in the wrong structure fields (for example, using ACTIVEWINDOW instead of ACTIVATE). To alleviate this problem, 2.0 has introduced modern synonyms that are less confusing than the old ones. For example, IDCMP_ACTIVEWINDOW and WFLG_ACTIVATE. This particular example of confusion (there are several) was the nastiest, since IDCMP_ACTIVEWINDOW when stuffed into NewWindow.Flags by accident corresponds numerically to WFLG_NW_EXTENDED, which informs Intuition that the NewWindow structure is immediately followed by a TagList, which isn't there! Intuition does some validation on the tag-list pointer, in order to partially compensate.

☐ Do not place spaces into the StringInfo->Buffer of a LONGINT string gadget. Under 1.3, it turned out that worked, but under 2.0, the validation routine that checks for illegal keystrokes looks at the contents for illegal (i.e. non-numeric) characters, and if any are found assumes that the user typed an illegal keystroke. The user's only options may be shift-delete or Amiga-X. Use the correct justification instead.

☐ If you specify NULL for a font in an IntuiText, don't assume you'll get Topaz 8. Either explicitly supply the font you you need or be prepared to size accordingly. Otherwise, your rendering will be wrong, and the user will have to reset his Preferences just to make your software work right.

☐ Window borders are now drawn in screen's DetailPen and BlockPen rather than Window's. For best appearance, you should pass an SA_Pens array to OpenScreen(). This can be done in a backwards compatible manner with the ExtNewScreen structure and the NS_EXTENDED flag.

☐ The full window border width is now rendered into, although the widths themselves are unchanged.

☐ Window borders are filled upon activation AND inactivation.

☐ Window border rendering has changed significantly for 2.0. Note carefully that the border dimensions are unchanged from 1.x. (Look at window->BorderLeft/Top/Width/Height if you don't believe us!) If your gadget intersects the border area that was declared, but not rendered into until 2.0, a visual conflict may occur. If Intuition notices a gadget which is substantially in the border but not declared as such, it treats it as though it were (this is called "bordersniffing"). Never rely on Intuition to sniff these out for you; always declare them explicitly (see the Gadget Activation flags GACT_RIGHTBORDER etc.) See Intuition Gadgets and Window Borders in the Additional Information section at the end of this document.

Preferences

❑ Some old struct Preferences fields are now ignored by SetPrefs (for example FontHeight). SetPrefs also stops listening to the pointer fields as soon as a new-style pointer is passed to Intuition (new-style pointers can be taller or deeper).

❑ Preferences ViewX/YOffset only applies to the default mode. You cannot use these fields to move the position of all modes.

❑ Preferences LACEWB bit lies when Workbench is in a new display mode (akin to GetScreenData()).

Workbench

❑ New screen sizes, screen top/left offsets, depths, modes, fonts.

❑ Default Tool now searches paths.

❑ New Look (boxed) icons take more space.

❑ Do not use icons which have more PlanePick bits set than planes (one IFF-to-Icon utility does this). Such icons will appear trashed on deeper Workbenches.

❑ New Look colors have black and white swapped (as compared to 1.3)

❑ The Workbench screen may not be open at startup-sequence time until some output occurs to the initial shell window. This can break startup-sequence-started games that think they can steal WB's screen bitplanes. Do not steal the WB screen's planes. (For compatibility, booting off pre-2.0 disks forces the initial screen open. This is not guaranteed to remain in the system). Use startup code that can detach when RUN (such as cback.o) and use CloseWorkbench() to regain the screen's memory.

❑ Under 1.3 the Workbench Screen and initial CLI opened before the first line in s:startup-sequence. Some naughty programmers, in an attempt to recover memory, would search for the bitplane pointers and appropriate the memory for their own use. This behavior is highly unsafe.

❑ By default 2.0 opens the initial CLI on the first _output_ from the s:startup-sequence. This allows screen modes & other parameters to be set before the user sees the screen. However, this broke so many programs that we put in the ''silent-startup'' hack. A disk installed with 1.3 install opens the screen as before. A disk installed under 2.0 opens silently.

☐ Never steal the Workbench bitplanes. You don't know where they are, how big they are, what format they may be in, or even if they are allocated. Recovering the memory is a bit tricky. We'll talk about the details at the DevCon talk. For the moment:

Under 2.0: Simply avoid _any_ output from your s:startup-sequence. If your program opens a screen it will be the first screen the user ever sees. Note that if ENDCLI is ever hit, the screen will pop open. You can avoid ENDCLI or use the techniques we'll mention at the talk.

Under 1.3: After ENDCLI, call the CloseWorkbench() function to close the screen. This also works under 2.0. Loop on CloseWorkbench() with a delay between loops. Continue looping until CloseWorkbench() succeeds or too much time has passed.

Layers

☐ Use NewLayerInfo() to create, not FattenLayerInfo,ThinLayerInfo,InitLayers.

☐ Simple-refresh preserves all of the pixels it can. Sizing a SIMPLE_REFRESH window no longer clears the whole window.

☐ Speed of layer operations is different. Don't depend on layer operations to finish before or after other asynchronous actions.

Graphics

☐ Do not rely on the order of copperlist instructions. For example, 2.0's MrgCop() builds different copperlists to that of 1.3, by including new registers in the list (eg MOVE xxxx,DIWHIGH). This changes the positions of the other instructions. We know of one game that 'assumes' the BPLxPTRs would be at a certain offset in the copperlist, and that is now broken on machines running 2.0 with the new Denise chip.

☐ Graphics and layers functions which use the blitter generally return after STARTING the final blit. If you are mixing graphics rendering calls and processor access of the same memory, you must WaitBlit() before touching (or deallocating) the source or destination memory with the processor. For example, the Text() function was sped up for 2.0, causing some programs to trash partial lines of text of their text.

☐ ColorMap structure is bigger. Must use GetColorMap() to create.

☐ Blitter rtns decide ascend/descend on 1st plane only.

☐ Some flying mode changes may break.

☐ GfxBase DisplayFlags and row/cols may not match Workbench screen.

☐ *Do not* hardcode modulo values - use BitMap->BytesPerRow.

☐ If the graphics autodocs say that you need a TmpRas of a certain size for some functions, then you MUST make that the minimum size. In some cases, before 2.0, you may have gotten away with using a smaller TmpRas with some functions (for example Flood). To be more robust, Graphics now checks the TmpRas size and will fail the function call if the TmpRas is too small.

☐ ECS chips under 2.0 use a different method of generating displays. The display window registers now control DMA.

Fonts

☐ Some font format changes (old format supported).

☐ Private format of .font files has changed (use FixFonts to create).

☐ Default fonts may be larger, proportional.

☐ Topaz is now sans-serif

☐ Any size font will be created via scaling as long as TextAttr.Flags FPF_DESIGNED bit is *not* set. If you were asking for some goofy size, like size 1 to get smallest available, or 999 to get largest available, you will get a big (or very very small) surprise now.

☐ Do *not* use -1 for TextAttr.Flags or Styles, nor as the flags for AvailFonts (one high bit now causes AvailFonts to return different structures). Only set what you know you want. A kludge has been added to the OS to protect applications which currently pass -1 for AvailFonts flags.

CLI / Shell

☐ Many more commands are now built-in (no longer in C:). This can break installation scripts that copy C:commandname, and programs that try to Lock() or Open() C:commandname to check for the command's existence.

☐ The limit of 20 CLI processes gone, and the DOSBase CLI table has changed to accomodate this. Under V36 and higher, use new 2.0 functions rather than accessing the CLI table directly.

☐ Shell windows now have Close Gadgets. The EOF char is passed for the Close Gadget of a Shell. This is -1L with CON: getchar, and the Close Gadget raw event ESC seq with RAW:.

☐ Shells now use the simple-refresh character-mapped console (see Console notes).

Console

☐ By default, CON: now opens SIMPLE_REFRESH windows using the V36/V37 console character mapped mode. Because of some differences between character mapped consoles, and SMART_REFRESH non-mapped consoles, this may cause incompatibilities with some applications. For example, the Amiga private sequences to set left/top offset, and set line/page length behave differently in character mapped console windows. The only known work-around is to recompile asking for a CON: (or RAW:) window using the SMART flag.

☐ Simple refresh/character mapped console windows now support the ability to highlight, and COPY text with the mouse. This feature, as well as PASTING text should be transparent to programs which use CON: for console input, and output. PASTED text will appear in your input stream as if the user had typed it.

☐ While CONCLIP (see s:startup-sequence) is running, programs may receive "<CSI>0 v" in their input stream indicating the user wants to paste text from the clipboard. This shouldn't cause any problems for programs which parse correctly (however we know that it does; the most common problems are outputting the sequence, or confusing it with another sequence like that for FKEY 1 which is "<CSI>0~").

☐ The console.device now renders a ghosted cursor in inactive console windows (both SMART_REFRESH, and SIMPLE_REFRESH with character maps). Therefore rendering over the console's cursor with graphics.library calls can TRASH the cursor; if you must do this, first turn off the cursor.

☐ Some degree of unofficial support has been put in for programs which use SMART_REFRESH console windows, and use graphics.library calls mixed with console.device sequences to scroll, draw text, clear, etc. This is *not* support in SIMPLE_REFRESH windows with character maps, and is being strongly discouraged in all cases.

☐ Closing an intuition window BEFORE closing the attached console.device use to work; it will now crash, or hang the machine.

☐ Under 1.2-1.3, vacated portions of a console window (e.g., areas vacated because of a clear, or a scroll) where filled in with the character cell color. As of V36 this is no longer true; vacated areas are filled in with the GLOBAL background color which can be set using the SGR sequence "<ESC>[>##m" where ## is a value between 0-7. In order to completely set the background color under V36/V37 send the SGR to set background color, and a FormFeed to clear the screen.

Note that SIMPLE_REFRESH character mapped consoles are immediately redrawn with the global background color when changed - this is not possible with SMART_REFRESH windows.

Additional Information

Task Switching

1.3 Kickstart contained two task switching bugs. After an interrupt a task could loose the CPU to another equal priority task, even if the first task's time was not up. The second bug allowed a task who's time was up to hold on to the CPU either forever, or until a higher priority task was scheduled. Two busy-waiting tasks at high priority would never share the CPU. Input.device running at priority 20 usually masked the effect of these bugs low priority tasks. Because of the bugs the ExecBase->Quantum field had little effect.

For 2.0 a task runs until either its Quantum is up, or a higher priority task preempts it. When the Quantum time is up, the task will now loose the CPU. Quantum was set to 16/60 second for 1.3, and 4/60 second for 2.0.

In general, the 2.0 change makes the system more efficient by eliminating unnecessary task switches on interrupt-busy systems (for example, during serial input). However, the change has caused problems for some programs that use two tasks of equal priority, one busy-waiting and one Wait()ing on events such as serial input. Previously, each incoming serial character interrupt would cause task context switch allowing the event-handling task to run immediately. Under 2.0 the two tasks share the processor fairly.

Intuition Gadgets and Window Borders

If 2.0 Intuition finds a gadget whose hit area (gadget Left/Top/ Width/Height) is substantially inside the border, it will be treated as though it was declared in the border. This is called "bordersniffing". Gadgets declared as being in the border or detected by Intuition as being in the border are refreshed each time after the border is refreshed, and thus aren't clobbered.

Special cases of note:

1) A gadget that has several pixels not in the border is not bordersniffed. An example would be an 18-pixel high gadget in the bottom border of a SIZEBBOTTOM window. About half the gadget will be clobbered by the border rendering.

2) A gadget that is not substantially in the border but has imagery that extends into the border cannot be sniffed out by Intuition.

3) A gadget that is substantially in the border but has imagery that extends into the main part of the window will be sniffed out as a border gadget, and this could change the refreshing results. A

common trick to put imagery in a window is to put a 1x1 or 0x0 dummy gadget at window location (0,0) and attache the window imagery to it. To support this, Intuition will never bordersniff such a gadget.

All these cases can be fixed by setting the appropriate GACT_xxxBORDER gadget Activation flag.

4) In rare cases, buttons rendered with Border structures and JAM1 text may appear invisible under 2.0. We apologize, but there is nothing that can be done on our end, even if the application technically did nothing wrong.

Intuition Fonts

The following table shows where the Intuition fonts come from.

	What you tell OpenScreen	Screen's Font	Windows' RPort's Font
A.	NewScreen.Font = myfont	myfont	myfont
B.	NewScreen.Font = NULL	GfxBase->DefaultFont	GfxBase->DefaultFont
C.	{SA_Font, myfont}	myfont	myfont
D.	{SA_SysFont, 0}	GfxBase->DefaultFont	GfxBase->DefaultFont
E.	{SA_SysFont, 1}	Font Prefs Screen text	GfxBase->DefaultFont

Notes:

A and B are the options that existed in releases prior to V36.

C and D are new V36 tags that are equivalent to A and B respectively.

E is a NEW option for V36. The Workbench screen uses this option.

GfxBase->DefaultFont will always be monospace. This is the "System Default Text" from Font Preferences.

The "Screen Text" choice from Font Preferences can be monospace or proportional.

'myfont' can be any font of the programmer's choosing, including a proportional one. This is true under all releases of the OS.

The menu bar, window titles, menu-items, and the contents of a string gadget use the screen's font. The font used for menu items can be overridden in the item's IntuiText structure. Under V36 and higher, the font used in a string gadget can be overridden through the StringExtend structure. The font of the menu bar and window titles cannot be overridden. Because the 2.0 Workbench screen uses option E to specify its Screen font from the user's Screen font Preferences, applications which open windows on the Workbench screen may get very large or proportional fonts in their menu bars, window titles, menu-items and string gadgets.

To predict your window's titlebar height before you call OpenWindow():

```
   topborder = screen->WBorTop + screen->Font->ta_YSize + 1
```

The screen's font may not legally be changed after a screen is opened.

IntuiText rendered into a window (either through PrintIText() or as a gadget's GadgetText) defaults to the Window RastPort font, but can be overridden using its ITextFont field. Text rendered with the Text() graphics.library call appears in the Window RastPort font.

The Window's RPort's font shown above is the _initial_ font that Intuition sets for you in your window's RastPort. It is legal to change that subsequently with SetFont().

ARexx

by Chris Ludwig

ARexx meets several important goals. It can provide a generic ''macro'' system which allows users to record their actions in an application and then ''play them back'' later. Probably most importantly, ARexx allows this macro capability to be extended across application in a standard way. Applications may ''control'' one another, and users may write ARexx programs which control multiple applications. Users at all levels can use ARexx to empower themselves and increase their personal productivity.

How ARexx grew to fulfill these goals is a story for another time, but be sure to stop William Hawes for some interesting stories, and to congratulate him on his important work for the Amiga community.

Implementing ARexx Support in Your Application

''What's in it For Me?'' Given the power of ARexx, it would be natural to assume that implementing support for it might be fairly difficult. Of course time and energy always seem to be in demand. These two facts might cause some developers to ask, ''Why bother?'' when it comes to adding ARexx functionality to their applications.

Well, first, it's important to realize that implementing ARexx support in an application is actually *much* easier than you might think. Thanks to ARexx being ''blessed'' by Commodore as an important part of the operating system, low level support routines actually come with any machine that runs release 2.0 or greater of the Amiga OS, and ARexx is also very much available for users who are still running under 1.3.

In addition to Commodore's support of ARexx, a number of public domain, freeware, and shareware products exist whose purpose is to facilitate the addition of ARexx to new or existing code.

Now, given the relative ease of adding at least minimal support for ARexx into an application, the next question is quite simply, ''Why bother?'' The answers to this question are numerous, and some of them vary simply by what kind of a developer you are and what kind of application you're developing.

One of the basic answers to "Why bother?" is, "Because users want it." Granted, there are probably more than a few users out there who would have trouble explaining *why* they want or need ARexx support in their applications, but to the marketing department, it doesn't really matter. The mere fact that they do want it should cause you to add it. The *why* behind a user's desire for ARexx doesn't come into play until after you've decided you're going to add ARexx support. At that point, you'll want to know WHY so that you can add ARexx in a way that will allow users to get done what they want to get done. It should be noted, however, that, as developers, it's not up to us to decide that we know *exactly* why a user wants ARexx--because one of the founding ideas behind ARexx for the Amiga is that developers can not meet *everyone's* needs in one application, but that providing ARexx support lets users *add in* the features that they want.

So here we have another reason to add ARexx into your application--It increases the immediate usefulness (and hence market share) of the application. There are many users out there (and in here ;-) who buy an application, and *immediately* sit down and customize it to suit their needs. I know that, given a set of applications that came "real close" to what I wanted and one that wasn't at all close--but would allow me to make it "exactly" what I want (through user-configurability and ARexx support), I would choose the configurable ARexxable product any day.

This leads to another important impetus for ARexx support. ARexx *extends* the usable lifetime of your product. Features that aren't present (or even conceived of) today can be hooked in tomorrow with ARexx.

Using the low level support provided by *Rexxsyslib.library* (found in the LIBS: logical assign of release 2.0 or greater) is relatively painless, saves you time, and most importantly, maintains compatibility. ARexx obviously is a high level language, and as such no one has ever suggested it as a replacement for assembly code. The fact that you might be able to think of easier or faster ways to generate some of ARexx's low-level structures might cause you to "roll your own" routines to generate them. Don't. There are parts of these structures which are "calculated" from the rest of the structure. The method used to calculate a hash code, for example, might change in the future, and your code might then cause ARexx to barf. You should only use anything other than *rexxsyslib.library* to build ARexx structures under very special circumstances (which we'll cover later).

Of course, there are even easier ways to add ARexx support to your code. Numerous public domain utilities exist that make it a snap to add ARexx awareness. There is the simple, yet elegant "SimpleRexx" by Mike Sinz. There is also the easy to use and powerful "MinRexx" (which is a good solution for adding ARexx functionality to existing programs) from Radical Eye Software, which is included with the ARexx developer's pack. The next step up is to use *RexxApp.library*, which is by Jeff Glatt and available on Fred Fish #463. *RexxApp.library* is quite simply a shared library version of the MinRexx code. This means that your application will be smaller, and that if there are any other applications using the library instead of *MinRexx.c*, even less memory gets used.

Using the Style Guide's ARexx Standards

The *Amiga User Interface Style Guide* includes an entire chapter which is devoted to ARexx. This chapter discusses a number of important aspects of standardization as they apply to implementing ARexx support in your application.

One of the important areas of ARexx implementation that has been standardized by the *Style Guide* is Port Naming. The name that your application gives to a message port that will be used with ARexx should follow the rules of the Guide. Briefly, the name should be ''<basename>.<slot number>'' where basename is the name of your application's executable (although the user should be able to change the base name) and <slot number> is the next available ARexx port slot for the application. Each invocation of an application should get a new slot number, as should each independent project which is simultaneously open. The user should be able to find out the portname of the current project by simply choosing the ''About...'' menu item.

Another area standardized by the *Style Guide* is a means of executing commands. The style guide recommends allowing users to open either a ''shell-style'' interactive console window or a requester that looks like Workbench's ''Execute Command...'' menu item's requester. Both of these methods work well. You might also consider using an ASL file requester to allow the user to choose an ARexx macro to execute.

Last, but certainly not least, the commands that your application accepts from ARexx and other applications should follow the standards set in the *Style Guide.* There is absolutely no excuse for using ''EXIT'' as a command to leave the program when ''QUIT'' is used by everyone else. This will allow users to quickly adapt scripts that they have written to new applications, and makes it very easy for you as a developer to know what to call a particular command.

Examples of Some Excellent Implementations

The Oxxi product *TurboText* shines as a good implementation of an ARexx-aware application. It is so intertwined with ARexx that it actually uses ARexx for some of its features (notably, some of its emulations rely on ARexx for extra functionality).

AmigaVision demonstrates that even the barest support for ARexx can greatly enhance an application--often in ways the original developer might not have guessed. For example, *AmigaVision* doesn't support the Toaster, but with the help of ARexx, it can.

Included here is the source to a program called *mouse* that opens an ARexx port and accepts commands which cause it to move the Intuition mouse pointer. This tiny application does just one thing, but it does it well, and can be quite useful when writing scripts whose purpose is to ''demonstrate'' something. It can also be useful for developers who are planning an *AmigaVision* based CDTV application, and have arranged to license ARexx. In many cases, a simple ''atomic'' program like *mouse* can be quite handy.

Mouse was written in C, and with a some help from ''SimpleRexx'' it has a nice small set of commands and follows all the rules.

Conclusion (Gee this is simple)

Your ARexx implementation can be the feature that sets your application apart from others, so it pays to do it and to do it well.

```
/*
 * mouse.c v1.0 by Chris Ludwig
 */

#include      <exec/types.h>

#include      <libraries/dos.h>
#include      <libraries/dosextens.h>

#include      <devices/input.h>
#include      <devices/inputevent.h>

#include      <proto/exec.h>
#include      <proto/dos.h>

#include      <rexx/storage.h>
#include      <rexx/rxslib.h>

#include      <stdio.h>
#include      <string.h>

#include      "SimpleRexx.h"


/* input.device stuff */
struct MsgPort *inputdevport;
struct InputEvent phony;
struct IOStdReq *inputrequest;
BYTE inputopenerror;

AREXXCONTEXT  RexxStuff;

void Quit(char whytext[],UBYTE level)
{
 if (!inputopenerror) CloseDevice((struct IORequest *) inputrequest);

 if (inputrequest) DeleteStdIO(inputrequest);

 if (inputdevport) DeletePort(inputdevport);

 FreeARexx(RexxStuff);

 printf("%s\n",whytext);
 Exit(level);
}


/*
 * Lattice control-c stop...
 */
int CXBRK(void) { return(0); }  /* Disable Lattice CTRL/C handling */
int chkabort(void) { return(0); }  /* really */


void click(UWORD code)
{
printf("in click()\n");

 phony.ie_NextEvent = NULL;
 phony.ie_Class = IECLASS_RAWMOUSE;
 phony.ie_TimeStamp.tv_secs = 0;
 phony.ie_TimeStamp.tv_micro = 0;
 phony.ie_Code = code;                /* button down */
 phony.ie_Qualifier = 0;
 phony.ie_X = 0;
 phony.ie_Y = 0;

 inputrequest -> io_Command = IND_WRITEEVENT;
 inputrequest -> io_Flags = 0;
 inputrequest -> io_Length = sizeof(struct InputEvent);
 inputrequest -> io_Data = (APTR)&phony;
```

```
   DoIO(inputrequest);


   phony.ie_NextEvent = NULL;
   phony.ie_Class = IECLASS_RAWMOUSE;
   phony.ie_TimeStamp.tv_secs = 0;
   phony.ie_TimeStamp.tv_micro = 0;
   phony.ie_Code = code | IECODE_UP_PREFIX;/* button up */
   phony.ie_Qualifier = 0;
   phony.ie_X = 0;
   phony.ie_Y = 0;

   inputrequest -> io_Command = IND_WRITEEVENT;
   inputrequest -> io_Flags = 0;
   inputrequest -> io_Length = sizeof(struct InputEvent);
   inputrequest -> io_Data = (APTR)&phony;

  DoIO(inputrequest);
 }


 void move(LONG mousex, LONG mousey)
 {
   /* send phony mouse message to input stream */

   phony.ie_NextEvent = NULL;
   phony.ie_Class = IECLASS_POINTERPOS;
   phony.ie_TimeStamp.tv_secs = 0;
   phony.ie_TimeStamp.tv_micro = 0;
   phony.ie_Code = 0;
   phony.ie_Qualifier = 0;
   phony.ie_X = mousex;
   phony.ie_Y = mousey;

   inputrequest -> io_Command = IND_WRITEEVENT;
   inputrequest -> io_Flags = 0;
   inputrequest -> io_Length = sizeof(struct InputEvent);
   inputrequest -> io_Data = (APTR)&phony;

  DoIO(inputrequest);
 }


/*
 * A *VERY* simple and simple-minded example of using the SimpleRexx.c code.
 *
 *
 * Note: You will want to RUN this program or have another shell available such
 *       that you can still have access to ARexx...
 */
void main(int argc,char *argv[])
{
short   loopflag=TRUE;
ULONG   signals;

int x, y, length;

  if ((inputdevport=CreatePort(0,0)) == NULL)
        Quit("Couldn't create a port for input.device",25);


  if ((inputrequest=CreateStdIO(inputdevport)) == NULL)
        Quit("Couldn't create request block for input device",25);


  if ((inputopenerror=OpenDevice("input.device",0,inputrequest,0)) != 0)
        Quit("Couldn't open input.device",25);

  /*
   * Note that SimpleRexx is set up such that you do not
   * need to check for an error to initialize your REXX port
   * This is so your application could run without REXX...
   */
  RexxStuff=InitARexx("mouse","mouse");

  if (argc)
  {
        if (RexxStuff) printf("Send commands to port %s\n",ARexxName(RexxStuff));
        else printf("ARexx is not available\n");
  }

  while (loopflag)
  {
        signals=ARexxSignal(RexxStuff);

        if (signals)
        {
              struct      RexxMsg*rmsg;

              signals=Wait(signals);
              /*
               * Process the ARexx messages...
               */
              while (rmsg=GetARexxMsg(RexxStuff))
              {
              char  cBuf[24];
              char  *nextchar;
              char  *error=NULL;
              char  *result=NULL;
              long  errlevel=0;

                    nextchar=stptok(ARG0(rmsg),cBuf,24," ,");
                    if (*nextchar) nextchar++;

                    if (!stricmp("CLICK",cBuf))
                    {
                              nextchar=stptok(nextchar,cBuf,24," ,");
                              if (*nextchar) nextchar++;

                              if (!stricmp("LEFT",cBuf))
                              {
                              click(IECODE_LBUTTON);
                              }
                              else if (!stricmp("MIDDLE",cBuf))
                              {
                              click(IECODE_MBUTTON);
                              }
                              else if (!stricmp("RIGHT",cBuf))
                              {
                              click(IECODE_RBUTTON);
                              }
                              else
                              {
                              /* Default to left button */
                              click(IECODE_LBUTTON);
                              }
                    }
                    else if (!stricmp("MOVE",cBuf))
                    {
                              nextchar=stptok(nextchar,cBuf,24," ,");
                              if (*nextchar) nextchar++;

                              length=stcd_i(cBuf,&x);

                              nextchar=stptok(nextchar,cBuf,24," ,");
                              if (*nextchar) nextchar++;

                              length=stcd_i(cBuf,&y);


                              move((LONG) x,(LONG) y);
                    }
                    else if (!stricmp("VERSION",cBuf))
                    {
                              result="mouse v1.0";
                    }
                    else if (!stricmp("QUIT",cBuf))
                    {
                              loopflag=FALSE;
                    }
                    else
                    {
                              error="Unknown command";
                              errlevel=20;
                    }

                    if (error)
                    {
                              SetARexxLastError(RexxStuff,rmsg,error);
                    }
                    ReplyARexxMsg(RexxStuff,rmsg,result,errlevel);
             }

       }
       else loopflag=FALSE;
 }
 Quit("QUIT command received.\n",0);
}
```

4th European Amiga Developers Conference
Milan, 10-14 September 1991

Multilingual Speech Synthesis: Communication and Education in the Global Village.

Prof. Raffaele Cocchi

In the beginning was the word, and it was spoken by the Amiga. As far as I know, the Amiga is the first popular PC with a speech synthesis capability incorporated. I bought an Amiga not for its amazing graphic capabilities, but for its amazing speech and graphic capabilities.

Marshall MacLuhan interpreted what was happening around us and to us after the invention of radio and TV, and talked about our world in terms of a global village.
Donald M. Lowe, in his History of Bourgeois Perception (Chicago, The Univ. of Chicago Press, 1982), moving from Middle Ages on, defines historical periods in terms of communications media, sensing hierarchy, and epistemic order, which applied to the Twentieth Century are "Electronics over typography", the extrapolation of sight/sound, synchronic system" respectively. I am sure that if Lowe was going to update his book, he would be compelled to state that this last quater of a century seems to exhibit a tendency to extrapolate sight from all the other senses, and put it right on top of that hierarchy.

I am not against the wonders of graphics and animation, which amaze me as much as they do you, I am only scared when this becomes our overwhelming preoccupation, our specialization. If we dedicate so much of our energies to sight, we have not enough left for the other senses. I usually have problems when I do not act as a whole, when my multitasking system shows bugs.

All this is just to tell you what everybody has always known, i.e., that right from the beginning the Amiga was the machine that could also speak, it was the perfect audio-visual tool for teaching anything, in schools especially. Why that did not happen and whose fault it was, however, is not the object of my talk here.

****

I have always been interested in language/s, because the only way in which I can know people is to compare what they say to what they do.
The kind of language I have been most interested in is literature, poetry especially, because up to this century it was considered something different from normal speech/writing. It was the best tool available for playing with words, structures, and meanings.
The best player were the poets because they not only aimed at deep meanings and imagination, but also at rhythm. "Good" poetry had to be a well balanced linguistic network, some kind of audio-visual text. It was the poets who first discovered "multimedia", working on synesthetic images, or texts appealing to all human senses.
Even Michelangelo, who was a sculptor, a painter, and also a poet, when admiring the beauty of his finished Moses, is said to have exclaimed: "Why don't you speak?" ("perché non parli?").

My opinion as an educator, is that the beauty of Amiga graphics made people forget that the voice synthesis was there. It's true that, beside kid games, there was a lot of early software like The Orator, Conversation with a computer, and Talker, and PD programs like Type and Tell. First Byte even adapted their speech synthesis to the Amiga and encapsulated it in a very limited talking text editor called SmoothTalker. For a few years we have also read in many Amiga magazines the advertisement for Señor Tutor, which was supposed to be a course for learning Spanish and the first Spanish speech synthesis, but nobody knows what happened to it.
The fact is that, except for the PD programs, none of the software I mentioned works on my OS 1.3/2.0 Amigas.
You can even find more than one course for teaching foreign languages with the Amiga, but they usually resort to the help of a recorded cassette for the pronunciation.

All Amiga users know that the SoftVoice speech synthesis licensed to Commodore is made of two pieces of software: the Narrator (device) and the Translator (library).
To describe the system in simple terms, we can say that:

1) the Narrator device consists of a series of modules containing a limited set of phonemes for the speech output of American English (for more information see the paper given by Mark Barton at the '88 US Commodore DevCon, "Inside Amiga's Translator and Narrator Software"), plus some linguistic rules.
To build a narrator is mainly a matter of competence in electronics and acoustics, though some general linguistic knowledge is needed.

The two main methods used in reproducing speech by synthesis are either diphones or formants. The Amiga Narrator uses the latter.
Before version 36.10 (7/30/1990 - 65488 bytes) of Wb 2.0 appeared, two other versions were "distributed" under the same code of 32.8 (11/30/1989), but with different lengths - 64308 and 65136 bytes respectively - probably two "beta" versions for the new OS.

2) The Translator library contains the set of grapheme-to-phoneme rules for American English.
In our case, to build a translator is a matter of linguistic competence only, because we can only access the ARPA phonetic symbols corresponding to the sounds (phonemes) offerred by the Narrator, in the form:

A=B(x,y) or x[A]y=B

meaning that "A=B" if "A" follows "x" and is followed by "y", and where "x" and "y" are variables which can be as complex as we can build.
All this means that anybody can build one or more translators, as I will demonstrate later, each for a different language or language variations like British, American, or Australian English.
Linguistic rules can be formulated or formalized in different ways, but are needed by any kind of speech synthesis, be it by diphones, formants, or other.
Though the Translator device has been minimally changed as a software, its rules are the same as those found in its first version and roughly correspond to the ones listed in Honey Sue Elovitz et.al, Automatic Translation of English Text to Phonetic Rules by Means of Letter-to-Sound Rules (Washington, DC: Naval Research Laboratory, U.S. Dept. of Commerce, NTIS, 1976), which are of public domain.

I am not going to talk about the Narrator because I have no specific competence whatsoever, and I would be compelled to tell you, in a very superficial way, something anybody can read in the technical literature.
Therefore I will limit my talk to the problems specific to translators, moving from SoftVoice's to other possible versions.

****

The level of usability of a speech synthesis device is mainly a matter of good and and bad speech, in the sense used by Daniel Jones (The Pronunciation of English, Cambridge: Cambridge Univ. Press, 1988) who says:
"'Good' speech may be defined as a way of speaking which is clearly intelligible to all ordinary people. 'Bad' speech is a way of talking which is difficult for most people to understand. It is caused by mumbling or lack of definiteness of utterance." (p. 4)

This means that it depends mainly on the sound quality of the speech synthesis, or the ability to (re)produce phonemes which are easily distinguishable from one another, allophones included (Narrator).
From this point of view I must say that SoftVoice did a wonderful job for American English, epecially when compared to the few expensive boards, usually for MS DOS PCs, I was able to test. To be absolutely truthful, DecTalk sounds better, but it costs twice as much as an

Amiga 3000.
Output sound quality is most important, but if we want a message to be clearly undestood by a listener, we have to take into consideration pronunciation and intonation, or text-to-speech rules as defined by the Translator.
In my opinion, the SoftVoice Translator does not score as well as their Narrator.
Last year, Dr. Eldo Bergman, a very active neurologist in the reabilitation of dislexic children, sent me from Houston (TX) a list of more than 8000 thousand words mispronounced by the translator - 50% of them are the kind of words everybody reads in their daily newspaper or listens to in TV programs.
Most commercial Amiga software uses the speech synthesis as it is. In a few cases, like Dr. Bergman's, the software contains a list of the words used transcribed in ARPA phonetic notation.

The most radical solution would be to rewrite the translator's rules from scratch, like I am trying to do with British English.
The most anti-economical solution for a professional software is to transcribe all the words. At the most this could be used in educational software for kids or in games, where the number of used words is very limited.
There is, however, an intermediate way to the solution, which lies in the concept of pattern matching so extensively used with wps for spelling checkers, thesauri, etc. To exemplify my statement, I will take 5 words from Dr. Bergman's list, each word followed by the Translator's automatic ARPA translation, the number of mistakes in the translation of each word, and Dr. Bergman's manual ARPA translation, that is the correct one.

a/band/-on	AE4BAENDUN	(2)	AXBAE4NDUN
a/band/-on-s	AE4BAENDAANS	(4)	AXBAE4NDUNZ
a/band/-on-ed	AE4BAENDOWND	(3)	AXBAE4NDUND
a/band/-on-ing	AE4BAENDOWNIHNX	(2)	AXBAE4NDUNIHNX
a/band/-on-ment	AE4BAENDAANMEHNT	(3)	AXBAE4NDUNMEHNT

As we can see, "abandon" (from the old French "à bandon") is the base word for the other 4, and all have two mistakes in common. The Translator doesn't have the following two pieces of information:

1) Though I do not know the origin of "abandon", and might be confused as to whether I should consider the initial "a*" a Saxon prefix meaning "on" (=on ban), or the initial "ab*" a Latin prefix meaning "from", in both cases "a" would be an unstressed "schwa", i.e., "a=AX".

The onfusion could be further compounded by the two possible syllabic divisions "a/band/on" and "ab/and/on".

2) According to the general English stress pattern, "the stress falls on the first syllable". If the first syllable is unstressed, I will be compelled to stress the second. The stressed "a" of my "'b/and", being followed by two consonants, is lax, i.e, we have "a=AE".

If I build an equivalence between my two patterns "aband*=AXBAE4ND*", I will eliminate the two mistakes which all the words have in common.
I might even be tempted to economize even further, and shorten the equivalence to

"aba*=AXBAE4*", but this would create problems with words like "a'base" or "a'bate", where the second "a" would become an "AE4" instead of an "EY4".

I am not going to consider the final "*on=UN(=AXN)" of "abandon", because it is correct.

I could go on like this explaining all the other mistakes, but our aim is not to give a course in linguistics, but rather to show what somebody without a specific linguistic competence, can do to solve the problem simply by plain observation.

In our case, we must:

a) check the word in a good pronouncing dictionary;
b) convert its phonetic notations into ARPA;
c) see that "abandon*" is always pronounced "AXBAE4NDUN" in all our words;
d) make the equivalency "abandon*=AXBAE4NDUN*", which leaves us with just one mistake out of the 14 possile.

If we could add the rule that final "*s" (marker of the possessive case, of the plural of substantives, and of the third person sing. of the present indicative) is pronounced "Z", after voiced consonants, except "s" and "z", we would correct also the last mistake, not only in the case of "abandons", but in any such case.

Before passing to the other pronunciation problems and to intonation, we should recall that we need some information about phonematics and phonetics if we intend to transcribe the dictionary's phonetic symbols into the ARPA ones.
Unfortunately not all dictionaries use the same phonetic symbols. The main discrepancies have always been between the American schools and the European IPA (International Phonetic Association), but they may soon disappear because of a real need for a standard.
A good book which explains this problem extensively is Geoffrey K. Pullum and William Ladusaw, Phonetic Symbol Guide (Chicago & London: The Univ. of Chicago Press, 1986).

The standard for phonetic transcription adopted in European projects for speech synthesis since 1987, and already applied to Danish, Dutch, English, French, German and Italian is SAMPA (Speech Assessment Methods Phonetic Alphabet).

To obtain a good pronunciation, though limited to American English, there are problems which are not easily dealt with by the simple grapheme-to-phoneme rules appearing in the Amiga Translator, e.g., the various kinds of homographs which need a minimum syntactic parser. Just to give two examples:
1) Homographs differing through a change in the pronunciation of the vowel, such as strong and weak verbal forms, for example: "have" = "/HAE4V" (verb) vs "have" = "(/H)AXV" (auxiliary).
2) Homographs differing through a change both in the pronunciation of the vowel and in the stress, for example: "concert" = "[THAX] KAO4NSERT" (noun) vs "concert" = "[TUW] KAXNSER4T" (verb).

Finally, we have intonation and prosody rules like pitch and pauses, which, in our case, are nicely controlled by the Narrator, but the lack of intonational difference between declarative

and exclamative statements could be considered a limitation.
The programmer/user can only change it by use of punctuation marks and stress markers.
Improvements could be also made by using a syntactical and semantical parser.

****

Up to now we have only addressed problems with American English, but we know that about 70% of Amiga users live outside the USA and speak a different language or different English dialects. I do not believe this silent majority would dislike a speech synthesis for their mother tongue.

We all know the main difficulty for adding a new English dialect or a new language translator to be used with the American Narrator, is the missing phonemes, which causes an American intonation in the best of cases, while in the worst ones we obtain such a dissimilarity from the language simulated, that a native speaker is almost incapable of recognizing it.
Old versions of the Narrator are more useful for simulating languages different from American English because they are richer in phonemes and allophones.

The undocumented phonemes, which are explained in a series of articles on speech synthesis (with reference to American English and German) by Prof. David Twigg-Flesner, published in AMIGA-MAGAZIN (August 1988) and Amiga Special (1988 Issues: 4/5, April/May; 6, June, and Extra1, July) and their corresponding ARPA phonetic symbols are:

- the aspirated allophones "/B", "/M", e "/R"
- "GH" e "GX", allophones of "G" -- "GX" is particularly useful after a stressed AA in order to avoid the exaggerated "whiplash-effect" produced by such combination.
- "KX" allophone of "K"
- "UX" allophone of "UW"
- "WH" for aspirated sounds in English words beginning with "wh*", like "who", "when", "where", etc.
- "NH" is useless because it usually induces in a system crash.

These additional phonemes are also included in the 36.10 version of Narrator, but they are not "transparent", i.e. they cannot be used - very probably they have been "de-activated", as appears to be the case with "/C".

Two other simple simulation problems are the following:

1) Vowels can be elongated by doubling or mixing them together, but there is no successful way of shortening them. Long vowels like AA and AO cannot be usefully doubled.
2) It is difficult to fight against the Narrator to obtaining allophones pre-set for American English according to their position in the phonetic chain, e.g, "tr*=CHR" in initial position as in "train".

****

We built 4 different Poly models between 1988 and last Spring. Passing from simple grapheme-to-phoneme rules like SoftVoice, to more complex ones which took into consideration a

simple but economic grammatical-syntactical parser, and the new rules resulting from my linguistic enquiry.
As I was compelled to simulate non-existent phonemes, my main object became a communicational one: What are the limits of the system in communicating an oral message that must be understood by a listener with a minimum training.
From an acoustical point of view, it meant I had to create linguistic systems containing phonetic oppositions such as to create the minimal amount of ambiguity in the listener, even though each phoneme may not be the right one.

The target I had set myself proved to be the limit of my models. My first practical scope was the application of the model to the teaching of foreign languages and literature. This meant I not only needed an acoustical output, but also a phonetic one.
ARPA is of no use in teaching foreign languages, but it's the only symbolic system allowing access to the phonemes of the Narrator.
At that point I thought it necessary to transform the ARPA output we obtained into an IPA one, and this created enormous complications.
The many ARPA allophones I created which needed to be translated, resulted in a great number of ambiguities to be solved. This tediously slowed down the execution of the code, and, worst of all, I was compelled to rewrite my rules every time I had a new Narrator.
The sense that we could not afford wasting time, forced me to finally awake from my long lethargy, and we decided to build this new model that translates my IPA rules into ARPA for the Narrator - a much easier task which almost allows me a 1 to 1 IPA-ARPA phonetic correspondence.

Our research attracted the attention of Prof. Zoppi (French chair at the Univ. of Turin), and we were kindly invited to participate in a CNR (National Council for Reasearch) strategic project which he coordinates on "translation in Europe".

From the beginning of 1991 this has meant:

1) Vanio Preti and I can work with other people who have competence we do not have.
2) The group has funds to buy hardware, software, and expensive books.
3) We are entering a specialized European scholarly network.
4) We can compare different hardware/software combinations with different points of view.
5) We can apply our growing linguistic knowledge base to the computer or boards that allow us the best results, because our rules are not "hardware dependent": they can be easily transformed (formalized in a different way) to meet different computer operating systems and programming languages.

The acquisition of Francesco Boschini, who is working in Prolog on the German rules for MS DOS with the assistance of Marcello Soffritti, greatly improved our ability to check the consistency of my Amiga rules, and in solving transportation strategies from MS DOS to Amiga, and the other way around.

Searching for one language rules is time consuming, doing it for 7 is a little more problematic. You can imagine how I grateful I am to Boschini, who is taking care of German now.

After the great model change, Preti created translation tools allowing me the minimum effort for adapting my old rules to the new model, and easy portability from one version to another of the Narrator.

At present, our polyglot model can deal with complex patterns which solve various kinds of homographic, syntactical, and grammatical ambiguities in the languages under observation, and general prosodic rules like the French liaison, or the so-called Italian syntactic reinforcement.

Poly is subdivided into three modules:

1) Stress deals with stress rules at word level.
2) Translate deals with grapheme-to-phoneme rules.
3) Prosody deals with stress rules at sentence level, but is still in its embryonic form.

This is the somewhat cyclic process we follow in our research:

1) The basic linguistic information found for each language is formalized into a corpus of rules.
2) The rules are compiled by PolyCompiler, a very efficient tool Preti created for permitting two kinds of tests:
a) an analytic one which shows all the failures to reach the target rules, thus allowing control on the economy of the path followed to reach each target;
b) a synthetic one which shows the target rules only, for verification of the hierarchical structure of the rules in the case of a failure;
c) a crosscheck of the two tests allows verification of the missing rules.
3) We create a dictionary file for each language extracting word lists from texts we elaborate with Letteratura Amica (L.A.) - a program similar to OCP (Oxford Concordance Package) or WordCruncher for MS DOS.
Word lists contain 15-35 thousand words, depending on the language.
4) The dictionary file is phonetically elaborated by Poly[glot] and transformed into an IPA phonetic word list file.
5) IPA phonetic dictionary files are manually checked against a pronouncing dictionary, corrected, and transformed in electronic pronouncing dictionaries.
This has been done only for French, Italian and German, and for the most frequent words only. The exception is British English for which I am using Daniel Jones, English Pronouncing Dictionary (London: Everyman, Dent, 1963, XII edition).
The MS DOS file was furnished to me by Oxford Text Archives. It was keyed-in by Prof. by Professor L. Guierre (Univ. de la Sorbonne, Paris - see Etudes de Linguistique Appliquée, 1966, Vol. 4, pp 48-64). The character set for pronunciation was rearranged by Prof. Roger Mitton (Dept. of Computer Science, Birkbeck College, London), and I adapted it to the Amiga.
6) The automatic IPA word list created by Poly is compared with the electronic pronouncing dictionary by Test&Tell, a program which gives us:
a) an output file containing all the mispronounced words with emphasis on the most typical failures;
b) the percentage of rightly and wrongly pronounced words.
7) The output file, with the help of the phonetic pattern search offered by L.A., is checked for rule refinement, i.e., for changing old rules or adding missing ones.

The languages I have tentatively simulated, in chronological order, are:

1) American Eng.- the one I started from when I decided to learn how to improve the Soft-Voice Translator.
2) Italian - I wanted to see how the Amiga could pronounce a foreign language I know sufficiently well.
3) Spanish - I wanted to the phonemes similar to the Italian ones.
4) German - for phoneme twin comparison English & German vs Italian & Spanish. Now I have handed the German rules over to Boschini and Soffritti, who are much more competent than me. I only solve problems of portability from their MS DOS model to our Amiga one.
5) French - for comparing the phonemes similar to the German ones, and for analyzing problems of nasalization.
6) Portuguese - for comparing its nasalized phonemes to the French ones.
7) British Eng. - This brings up us against the problem of dialects: British vs American English.

****

What is the use of our present foreign speech synthesis can be explained only in terms of the real applications we have made, are making, and are interested in making in a near future:

1) Education & research

Finished product: Letteratura Amica v. 2.03
Beside doing most of the things OCP and WordCruncher do, it can analyze texts from a phonetic point of view, i.e., it can observe patterns of speech and rhythm, or rhetorical speech devices, in texts written in different languages, when such observation is useful, e.g., in poetry, transcribed oral documents, advertisements, oratorical speeches like those given by people who are trying to convince an audience to buy something - people like TV salesmen and preachers, politicians, etc.

The words of the text can be pronounced out loud by the Amiga in different ways, i.e., the level of phonetic preciseness and its corresponding pronunciation depends on our grapheme-to-phoneme rules.
The actual set of rules implemented in L.A. are the ones linked to last finished Polyglot model. As soon as we complete our new model, rules will be changed and continuously updated.
As there is no set of formal rules capable of controlling a natural language, there will always be some pronunciation mistakes.
As we all know, any mistake can teach us something, so we allow the user to correct the word pronunciation through their own phonetic transcription. This also means that a good teacher can help students in learning the pronunciation of foreign languages, correcting the computer, i.e., the "bad" student is temporarily transformed into a good teacher, often avoiding being made fun of by the better students, which could cause him serious psychological problems. The computer becomes the scape-goat of the whole class.

The program is accompanied by a hyperhelp, at present available in English, German, and

Italian - French and Spanish will be finished in a month.
We are also completing an hyperbook containing an introduction, 5 tutorials, and an appendix. The text for the hyperbook is already available in English and Italian.

2) Presentations, education and handicaps

The major limits to our polyglot model for multilingual speech synthesis applied to conversion of texts into continuous speech are:
A) the lack of certain phonemes that cannot be efficiently simulated, which cause an foreign intonation effect unpleasant to the ear of the mothertongued listener;
B) a too limited prosodic patterning - specific to each language - causes an unpleasant and annoying monodic intonation.

I believe that Michele Console Battilana, a commercial developer well known to all of us, who is attending this conference, could easily tell you about these problems, as far as Italian is concerned. In fact, he developed indipendently an Italian translator incorporated in a program for the help of people with severe motorial handicaps.

If we were to build a PolyTranslator.device, our rules could easily be used by any Amiga software. The demonstration we are going to show you now, presents an intermediate model working with AmigaVision. This means we could display an Amiga 500 in a shop window to show presentations of various products on sale there - messages to be spoken can be typed and improved with any editor.
Software houses could create CAI and CALL courses for teaching languages without the help of a tape recorder; or courses for teaching and learning phonetics with the simple addition of a rule compiler. Also possible are telephone applications for office automation such as can be found for MS DOS, etc.
With Poly a teacher could even guess, for instance, what will be the most probable mistakes of a French student beginning to learn Spanish. Just read a Spanish text with the French translator.

I must admit, however, that at the present state of our art, only kids and handicapped people could really profit from such a device, i.e., human beings who really know what it means to play, and other who have to substitute bodily functions they lack and can give up some kind of perfection for a bit of independence and freedom.
People who do not have these kinds of problems are not very interested in solving them, unless it can bring them good money.
A lot of human factors research show that most people are opinionated and even annoyed by "machines that speak", even if they use the best quality on the market. It's the psychological factor: they are prepared to make enormous efforts trying to understand a person with a phonation handicap or a foreigner ravaging their mother tongue, but very little effort to try to understand a computer. People act while they are speaking, they move thier eyes and mouth, even their hands. They show emotions. While a computer is there in all his scientific rigidity.
At Speech Tech '90 in New York, I attended a lecture given by a brilliant speech synthesis engineer from Japan. The problem was that nobody could understand more than 10% of what he was reading in his pseudo-English. Two attendees sitting near me, were agreeing in a low voice that his text would have been better understood by the audience, if read by their speech board. I told them that my Amiga would have read that text in a cleare "voice".

Maybe everybody would have agreed with me, but nobody would have dared to propose it, and not just for the sake of politeness!.
Most ordinary people like us, who can see, would not pay much attention to "professional"speech coming out of a computer. We might even laugh at the best speech output, just because it sounds un-human. But there are people who are not in our lucky condition, people who have lost their sight and prefer to use hard to understand and costly speech boards for writing their papers and read texts they are interested in material sent to them, rather than depending on other people. They know it's bad speech, but they need it, and have adapted to it.
We the Amigos, the superficial seer of the wonders of animated 24 bits graphics, have a responsability to pass all those amazing images to them verbally, at least by means of decent speech synthesis.

Speech synthesis can solve problems of both visual and phonation handicaps, allow people who cannot speak to communicate with people who cannot see.
We are showing VoceAmica (friendly voice), a simple interface that can help people suffering those handicaps in using the Amiga.
It can be improved, and we intend to do it. It is my intention for it to become a polyspeaking Amiga manager/editor.

Which of the languages we are working on can be efficiently used by handicapped people? Though the answer should be yours, at present or in the very near future when our new Polyglot will be fully working, in addition to American and British English, I believe that only German, Italian, Portuguese, and Spanish could pass the exam. I have no hope whatsoever for really improving my French, even if I work hard at the prosody module.
These kinds of problems can be solved only by the implementation of the missing phonemes.

The problems with phonemes have been almost successfully solved by boards working with MS DOS computer like the ones produced by:
1) InfoVox (Sweden) for British and American English, French, German, Norwegian, Spanish, and Swedish. The board or box (+ a language) costs $ 2000 c.a - each additional language is $ 700 c.a;
2) Multivox (Hungary) for Esperanto, German, Hungarian, Italian, Spanish, Norwegian. The box (+ a language) costs $ 1200 - each additional language is $ 150-200 c.a
3) Olivetti for Italian and English. The board and the two languages cost $ 2000 c.a.

A good multilingual speech synthesis integrated into the Amiga OS would not only be useful for simple educational programs, or for thousands of games, but could be used for the CDTV. If it's true that a CD can hold an encyclopedia, we will have to wait for a long time for a CD capable of hold an encyclopedia, together with its text recorded by a human voice, no matter what the efficiency of the compression algorhythm.

The only economic way to obtain vocalic output of large corpuses of text is voice synthesis. Its use would mean that multilingual dictionaries, multilingual editions of the same literary work, and generic encyclopedic works or textual databases would be accessible to people suffering visual handicaps. It could be of great help for recovering dislexic children and others suffering from different phonation handicaps in conjunction with very simple software specifically designed, and would amaze all the children of the world, and not only in American words.

4th European Amiga Developers Conference
Milan, 10-14 September 1991

Commodore in Italy

Stefano Lamon
Application Marketing Manager
Commodore Italiana S.pA.

1. INTRODUCTION

The purpose of this document is to give some useful information to the Amiga developers who want to operate in the italian market. It is not meant to be an exhaustive exposition, but should anyway give to the foreign reader a better understanding of the italian market, with particular attention to the Commodore world.

2. COMMODORE ITALIANA

The italian subsidiary of Commodore International was founded in 1982, and was initially based in Cinisello Balsamo, a town in the Milan area. Afterwards, in 1989, it moved to a new office district called Tecnocity, within the city of Milan.

At the time being, it counts around 40 employees, in the Administrative, Marketing, Commercial and Technnical groups. The charge of Managing Director is covered since October 1989 by Werter Mambelli, recently appointed to Head of Southern European Region (Italy, Spain, Portugal).

The marketing activity is mainly carried on by Nicoletta Toniutti as Marketing Development Manager, Carlo Zambellini, in charge of Product Management, and the writer, as Application Manager. Piero Delorenzi is the Financial Controller, while the Commercial Group is splitted into two sections, called "Systems" and "Consumer", respectively led by Luciano Lini and Albino Bertoletti. The Support Manager is Moreno Zampolli.

3. THE ITALIAN MARKET

The italian firms were keeping a good growing trend until the end of 1989, especially those located in the north of the country. The information technology market is mainly developed in the Milan-Genoa-Turin area, with nearly 40% of the whole national business. Also the Bologna area as a high concentration of electronic industries.

The computer market, however, has not grown as expected in the last year. Infact, the forecasted growth of 20.8% for the 1990 has actually resulted in a growth of about 12% in value considering the small/medium companies, while big firms such as IBM and Olivetti reported an even worst figure.

A similar trend is likely to be registered for the first half of this year. Commodore is an exception, in this scenery, with a growth even bigger than the more optimistic forecasts: the company increased its total income of 27% over 1989, and the trend is remaining steady in the first half of 1991. Commodore Italy is on its way to the 100 million $ goal for the calendar year 1991.

Looking at the PC market (including home micros), the shipped quantity of 560.000 in 1989 has become 700.000 in 1990 (Source: Dataquest).

The Commodore installed base is divided, in Italy, as follows:

C64	around 3.000.000 units
A500	more than 400.000 units
A1000	around 18.000 units
A2000	around 20.000 units
A3000	around 1.000 units
Low-end PCs	more than 10.000 units
High-end PCs	around 1.000 units

It comes clearly out that Commodore Italy is keeping a good growing trend, comparing to the stagnation affecting the rest of the market. The above mentioned figures also denote Commodore's undisputed leadership in the consumer market, with its best sellers C64 and A500, and more and more aggressive entry on the systems business.

4. THE DISTRIBUTION NETWORK OF COMMODORE ITALIANA

Commodore products are distributed in Italy through three main channels: the "Commodore Points", the "Commodore System Dealers", and the wholesale distribution.

The Commodore Points are mainly operating in the consumer electronic business, and are spreaded through all the country, with a total of 450 points of sale that will become 500 by the end of this year. The Commodore Point are served directly by Commodore Italiana. Although most of these entities are mainly consumer oriented, some of them are technically skilled and therefore sell high-end PCs and the Amiga 3000 series.

The Commodore System Dealers, about 40, are more targeted to the corporate market, selling solutions based on the Amiga 3000 and high-end PCs. Tipically, the System Dealers are able to provide a full support service to the customers, including software and peripherals. The System Dealers are served directly by Commodore Italiana.
Wholesale distributors, instead, sell only consumer products to those points of sale that are not directly in contact with Commodore Italy.
Unlike other european countries, in Italy the business is mainly based on small/medium shops largely spreaded through the country; there are only a few chain and discount stores: the most important for the consumer electronic business are "Expert" and "Coeco".

5. SOFTWARE AND HARDWARE FOR WHICH MARKET?

As above mentioned, a large section of the installed base comes from the consumer area, where the Amiga 500 has the bigger share.
This market is made up mainly of home users, that bought the computer for entertainment and/or personal productivity, especially for those "creative" activities where the Amiga can give its best. Therefore, there is a strong request for recreational software, paint programs, 3D and animation packages, and music programs.
Also "serious" productivity software is getting a good acceptance, especially word processing and utility packages.
One application area where the Amiga has a significant advantage is desktop video: demand is always steady and increasing, as VCRs and camcorders get more and more diffused.
Speaking of professional applications, we believe the Amiga can praise a significant success in three fundamentals markets: video applications, multimedia, and creative graphics.
In the video market the Amiga is well known and is currently being used by the goverment's networks as well as many private TVs. Probably not everybody knows that in Italy there are nearly 900 private TVs, and at least 5000 production and post-production facilities.
Moreover, many packages intended for video applications may be easily adopted for corporate presentations. For all these potential customers, the Amiga can be the right solution for cost-effective titling, 2D and 3D animation, and video effects in general.
Multimedia market is still pretty new, and probably the most rapidly growing segments are those of points of information and unattended points of sale.
The introduction of the CDTV has also opened a new world of opportunities, from entertainment to informative and teaching packages.
The educational market is strongly conditioned by government's regulations, requiring MS-DOS compatibility. Commodore Italy sidestepped this obstacle providing A2000s with pre-installed Janus Bridgeboards and MS-DOS software, during a commercial operation started in 1988, and lasted for about two years.

6. TRADE FAIRS

The most important information technology show in Italy is undoubtfully SMAU, considered the second largest exhibition in Europe, after the giant annual trade fair CeBIT held in Hannover, Germany.
Running in the first week of October, in Milan, SMAU is the appointment that no firm significantly operating in Italy would miss.
Commodore has its istitutional booth since beginning of operations, covering last year a 900

square meters area, hosting several third parties with their applications.
Starting from last October, Commodore is also present at the International Broadcasting and Telecommunications Show (IBTS), with dedicated solutions for the video/audio markets.
While SMAU is open to the public, IBTS is restricted to the market operators: SMAU had in its last edition nearly 200.000 visitors, while IBTS counted around 10.000 presences.
It is common philosophy at Commodore Italy to involve third parties during these exhibitions: we welcome proposals in this sense from domestic and foreign developers.

7. SOFTWARE REVIEWS: SPECIALIZED MAGAZINES

It is of great importance, to achieve a good penetration of a market, to have one's products tested and reviewed by th press. Following are the most diffused magazines, divided by product/application field:

A) Magazines dedicated to Commodore and/or Amiga:

Commodore Gazette - IHT Gruppo Editoriale
Via Monte Napoleone, 9 - 20121 MILANO

Amiga Magazine - Gruppo Editoriale Jackson
Via Pola, 9 - 20124 MILANO

Enigma - GR Edizioni
Via XXV Aprile, 3 - 20097 San Donato Milanese (MI)

Amiga Byte - Arcadia
Corso Vittorio Emanuele, 15 - 20122 MILANO

B) Information technology magazines with inserts dedicated to Amiga:

MC Microcomputer - Technimedia
Via Carlo Perrier, 9 - 00157 ROMA

BIT - Gruppo Editoriale Jackson
Via Pola, 9 - 20124 MILANO

C) Games/entertainment magazines:

Zzap! - Edizioni Hobby
Via della Spiga, 20 - 20101 MILANO

Kappa - Glenat Italia
Via Ariberto, 24 - 20123 MILANO

The Games Machine - Edizioni Hobby
Via della Spiga, 20 - 20101 MILANO

D) Information technology magazines:

Micro & Personal Computer - Gruppo Editoriale Suono
Via Capo Peloro, 30 - 00141 ROMA

PC Magazine - Gruppo Editoriale Jackson
Via Pola, 9 - MILANO

PC World Italia - IDG Communications Italia
Via Vida, 7 - 20127 MILANO

Zerouno - Mondadori Informatica
Via Rivoltana, 8 - 20090 Segrate (MI)

Microcomputer & Software - Tecniche Nuove
Via Ciro Menotti, 14 - 20129 MILANO

Informatica Oggi - Gruppo Editoriale Jackson
Via Pola, 9 - 20124 MILANO

Future Office - Gruppo Editoriale JCE
Via Ferri, 6 - 20092 Cinisello Balsamo (MI)

E) Video and multimedia magazines:

Media Production - Gruppo Editoriale Giunti
Ripa di Porta Ticinese, 91 - 20143 MILANO

Monitor - Media Age
Via Stefano Jacini, 4 - 20121 MILANO

Millecanali - Gruppo Editoriale JCE
Via Ferri, 6 - 20092 Cinisello Balsamo (MI)

Computergrafica & Multimedia - Gruppo Editoriale Jackson
Via Pola, 9 - 20124 MILANO

F) Desktop publishing and graphics magazines:

Italia Publishers - Gruppo Editoriale JCE
Via Ferri,6 - 20092 Cinisello Balsamo (MI)

Creative - Edizione Creative
Via 1 Maggio, 25 - 20048 Oggiono (MI)

8. SOFTWARE AND PERIPHERALS DISTRIBUTORS

Following are the addressess of some distribution companies currently known to be operating in Italy in the Amiga market. This list is not meant to certify the therein mentioned companies, nor to disqualify other firms. Commodore Italy does not assume any responsibility on the accuracy and/or updating of these information.

A) Software

Applied Peripheral & Software
Via Giovanni XXIII - 33040 Corno di Rosazzo (UD)
Tel. +39 432 759264 - Fax +39 432 759264

Cloanto Italia s.a.s.
Via G.B. Bison 24 - 33100 Udine
Tel. +39 432 46612 - Fax +39 432 609051

C.T.O. s.r.l.
Via Piemonte 7/F - 40069 Zola Predosa (BO)
Tel. +39 51 753133 - Fax +39 51 753418

Leader Distribuzione s.r.l.
Via Mazzini 15 - 21020 Casciago (VA)
Tel. +39 332 212255 - Fax +39 332 212433

Softel s.a.s.
Via Alessandro Salinas 59/A - Roma
Tel. +39 6 7231811 - Fax +39 6 7231812

B) Add-on and peripherals

ACS
Via Cà Pirazzone 46/A - 47031 Domignano (Repubblica di San Marino)
Tel. +39 549 902500 - Fax +39 549 906312

Flopperia
Viale Montenero 15 - 20135 Milano
Tel +39 2 55180484

Newel
Via MacMahon 75 - 20155 Milano
Tel. +39 2 3270226

Non Stop Electronics
Via Buozzi 11 - 40057 Cadriano (BO)
Tel. +39 51 765299

Soundware
Via Mazzini 12 - 21020 Casciago (VA)
Tel. +39 332 223766 - Fax +39 332 228288

9. PIRACY AND COPYRIGHTS

Unathorized duplication of software is unfortunately diffused also in Italy, especially in the entertainment market.
The author's rights protection in Italy is ruled by a law created exactly 50 years ago, and this is currently the only existant reference. Nevertheless, it has to be said that the Italian Magistracy is currently following the direction of the rest of Europe, as demonstrated by the latest judgments in this matter.
For this sake, it may be useful to know that there is an organization in Italy called ASSOFT, member of Business Software Alliance, whose main goals are:

- to protect its members against any form of dishonesty and unfair competition
- to address the users toward those companies correctly operating in the market, ensuring genuineness of the products
- to promote by the competent authorities the adoption of laws and rules intended to protect the quality and genuineness of the products
- to promote and coordinate technical research for the phisycal protection of software
- to keep its members informed and up to date in respect to the evolution of italian and foreign laws

For those who need more information, here's ASSOFT address:

ASSOFT - Associazione Italiana per la tutela del software
Via Benvenuto Cellini, 1 - 20129 MILANO
Tel. +39 2 55183116-5400025
Fax +39 2 55012110

10. EXPORTING TO ITALY

Depending on whether you have to ship goods to Italy for demonstration or sale, temporarily or not, there are different procedures to be followed.

A) Temporary export:
The fastest way is the ATA Carnet. This document, released on request by the Chamber of Commerce (or equivalent organization) of the country where the goods come from, allows temporary import into Italy of any good, with the following restrictions:

- the goods must be recognizable (serial numbers or seals placed by customs at arrival in Italy)
- the goods must be returned back at the same country where they originally came from
- the goods returning back must be exactly the same which came in

For countries outside the EEC may be requested a caution money of around 5% of the value declared in the invoice. The ATA Carnet requires that the goods travel together with the owner; if this is not possible, the sender must ensure that the consignee takes care of customs clearing and subsequent return of the goods.

B) Definitive export for demonstration purposes:
In this case the goods must be accompanied by the T2 document for EEC countries, and if possible by the Certificate of Origin for the other countries. An import license may be necessary for goods coming from Japan. The pro-forma invoice, indispensable to clear through the customs, must declare the exact value of the goods. In case of software, it should be labeled as "DEMO - NOT FOR RESALE", in order to declare only the magnetic and paper support value. Consignee will have to pay V.A.T. (19%) on that value. If the goods come from non-EEC countries, an import duty of 5% of the total value may be requested.

C) Definitive export for sale:
Same as above, except for the invoice, which is not a pro-forma and should report the exact value of the goods. An important note: no italian company is allowed to transfer money to a foreign company without the so-called "proof of evidence". This means that if you want your goods to be paid by an italian company, you must always send your goods with an accompanying invoice, which has to be stamped by Italian Customs and finally get payed by the consignee. If you don't have a stamped invoice you will never get paid, because it's illegal.

D) Export for attempted sale with subsequent re-import of unsold items:
This particular form of export, used in other countries especially during fairs where exhibitors can sell their goods to visitors, is really difficult to comply with in Italy, due to the current fiscal rules. We strongly suggest you to avoid if possible this procedure, or otherwise refer to a forwarding agency.

4th European Amiga Developers Conference
Milan, 10 -14 September 1991

From Programming to Finished Product

Roberto Donelli
Soft Service Division Director
MEE S.p.A. - Milano

1. INTRODUCTION

To complete the development phase of a program does not mean to have finished a software product and to be ready to market it.

To place a program on the market, the product has to go through a number of production steps beyond the usual activities of a programmer, to transform it into a marketable package.

Those are the main phases:

- Realization of a Gold master
- Copy protection
- Realization and testing of a Silver master
- Duplication of the medium
- Quality control

After those steps, or in parallel with them, there will be the realization of manual, labels, packages and perhaps the localization of the product for different national markets, but we will not address those activities, mostly graphical, in this presentation.

Among the steps enumerated before, only the first two are directly under the responsibility of the programmer/development team, while the others, still under the supervision of the developer, have to be performed by a company specialized in software duplication.
The right choice of a software services company, who could really become a partner for the developer, may be of the utmost importance to save time and money.
A good duplicator must be able to perform all the steps of its concern in a production timely and with the utmost reliability while cooperating with the developer in all the other phases of the realization of the final product.

The goal of this presentation is to supply to a developer some clues on how to approach the problem of preparing a finished product and obtaining the best possible performance from a duplicator.

2. GOLD MASTER

The first operation that a developer has to address is to make a Gold master. A Gold master is the final copy of the floppies containing the product, made on brand new disks that would be formatted first and onto which the program(s) will be copied file by file. To conform to duplication standards it is not correct to realize a Gold master using a diskcopy or a track-by-track copier.
The Gold master must be supplied in two copies to the duplication center, either already copy protected, if the developer can make its own protection, or with instruction on if and how to install a copy protection scheme.

3. COPY PROTECTION

In the event that the copy protection has to be installed by the duplication center it is likely that it will be one of the several standard copy protection schemes and so more likely to be successfully attacked by copiers, thus making it less reliable.
Some duplication centers have an internal R&D center for custom copy protection, but this features will raise the costs and stretch the time to obtain the finished product.
The ideal choice, thus, would be for the programmer to develop its own copy protection, but to do so it is necessary to respect some rules.
First of all, we should keep in mind that a protection must not be reproducible, but not by everybody! A protection that cannot be duplicated by a good duplication center, being made empirically, is just a nice but in fact useless exercise.

It is necessary to know which duplication system is available at the duplication firm. There are currently only two machines that are able to mass-duplicate protected software with good reliability: Trace series 1000 (in decreasing order of performance: 1020, 1006, 10HT with multiformat upgrade) or a system based on a PC, IBM compatible, with a Ghost Writer II duplication board. The limitation of the latter system is the usage of predefined formats. The system reads a track, analyzes it comparing its structure with the known formats and, once identified the right format, it makes sure that the format from the master corresponds to the standard one. Then it computes EDCs and if everything is OK it validates the track and step to the next track. If on master disk there is something that is not part of the standard, then the system moves to blind mode and copies the track in a raw mode. This system, thus,

is not able to handle sophisticated protection schemes, and can be compared to the methods used by several well known copiers. In other words, a protection that can be copied by Ghost Writer can also be copied, maybe with less reliability, by a nibble copier.
Therefore, if we want to make and standardize a good copy protection scheme, is necessary to work with a duplication center that uses a more powerful system. Using Trace system the limit to what can be done is only the ability of the technician that makes the protection and that encodes it in a format file readable by the machine. The Trace system, in fact, is not bound by predefined formats. Before reading the master it loads a format file given by the operator and on that basis it will create, in its memory, an image of the disk. According to that image, it will read the master and verify the validity of the areas of sector reserved to the format and data, while computing and comparing the EDCs. If no errors are found the machine creates a duplication image; otherwise it stops, asking a better master or a new format file. So, if your master works, the result depends on the format file.

This file is written in a language called "Freeform" realized by Magnetic Design Corp. with the goal of facilitating the design and implementation of format specifications on Trace 10xx duplication systems. These format specifications are used to control the tasks of formatting, duplicating and verifying a floppy disk.
The formatting process will write on disk the information needed to identify the tracks and sectors available on the disk. These information can be used by the destination system to read and write data on the disk. While formatting is also possible to specify an initial setup for areas reserved to directory and boot data. Every good copy protection system is necessarily located in the ID and operating system areas of a sector, and thus it is possible to include the protection data in a format specification.
The duplication is the process of copying entire areas of data that are on the master disk, including the track and sector IDs. During this process there will be also the verification of the data just written.

All the format specifications must be encoded in the format file in non ambiguous way; otherwise the system would not be able to discriminate between the protection scheme and a real error that must be corrected before the duplication is completed.

The typical Freeform format file is split in three sections:
- Identification block: it contains filename, description, ID number and version number for the format specifications.
- Drive specifier: identifies the physical type of the drive that will be used to read and write the format and the name of the precompensation file to be used.
- Format description: it contains a full description of the format for each track of the disk. A minimal format description contains the declaration of the encoding method, the dimension (or length) of the bitcell and obviously the description of the track and sector structure of the disk.

All those information are specified through reserved keywords.
Using Freeform it thus possible to implement any copy protection scheme.

The methods most frequently used are a combination of the following:
- extra tracks
- extra sectors

- illegal track, sector, side, offset specifications
- incorrect EDC
- hidden sectors
- missing sectors
- only one sector per track
- flaky (or weak) bits
- additional sectors in data area
- illegal speed changes
- illegal marks
- narrow gaps
- wrong gap characters
- tracks or sectors not in Amiga format

The Appendix A is a sample of Amiga format, written in Freeform.

Lets try to understand better this format file.
An Amiga floppy is double sided, has 80 track per side (numbered from 0 to 79), 11 sector per track (numbered from 0 to 10), 512 bytes of data per sector. The total capacity, so, is ((2*80*11*512)/1024)=880 Kbytes of data.
The atomic element of this format definition is the sector.
In this chart you can see the form of a sector.

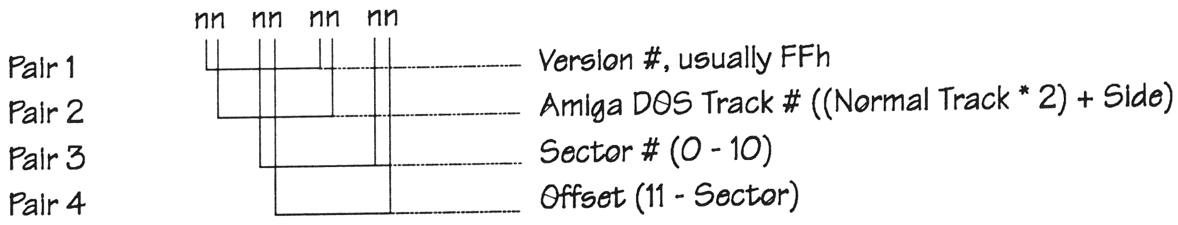
# of bytes	hex value	type	notes
150	00	gap	for sector 0 only
002	00	gap	inter-sector gap
002	A1	mark	clock 0A
001	FF	header	format type
001	xx	header	track #
001	xx	header	sector #
001	xx	header	offset
016	xx	data	operating system
004	xx	EDC	ID checksum XOR16
004	xx	EDC	Data checksum XOR16
512	xx	data	
xxx	00	gap	fill byte after last sector

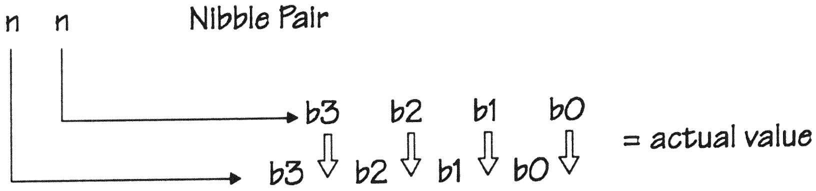
The Amiga format uses three kinds of encoding: AMG-1, 2, 3.
AMG-1 encoding, used for marks and gaps, is a standard MFM encoding.
AMG-2 encoding, used for the four header bytes, is quite peculiar. The four header bytes have to be treated as eight hex digits, grouped in four pairs. The first pair is made up by the first and fifth digit, the second by the second and sixth, the third by the third and seventh, and the fourth by the fourth and eighth.
To compute the actual values for the header, then, each digit has to be considered as a binary nibble and the two nibbles have to be interleaved each other.
(see diagram)

AMIGA HEADER (AMG-2)

Follows 2 x A1 (clock OA) - four bytes long

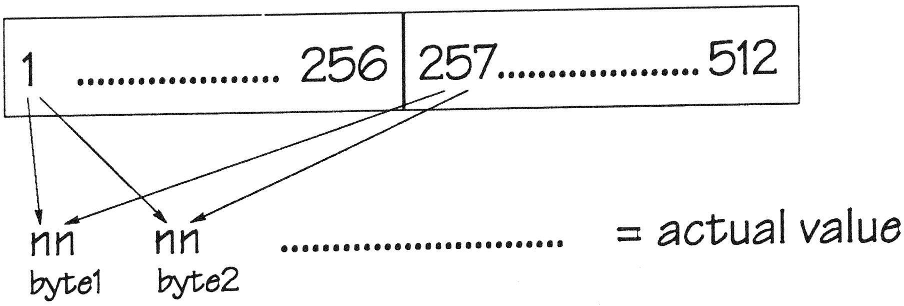
A1 A1 nn nn nn nn
OA OA


nn nn nn nn
Pair 1
Pair 2
Pair 3
Pair 4
Version #, usually FFh
Amiga DOS Track # ((Normal Track * 2) + Side)
Sector # (0 - 10)
Offset (11 - Sector)


n n
Nibble Pair
b3 b2 b1 b0
b3 b2 b1 b0
= actual value

AMG-3 encoding is used for data area: the 512 bytes of data should be imagined as split in two chunks; from byte 1 to 256, from 257 to 512. To obtain the actual value for those bytes we should pair the first nibble of the first byte with the first nibble of the 257th byte, the second nibble of the first byte with the second nibble of the 257th byte, the first nibble of the second byte with the first nibble of the 258th and so on.

AMIGA DATA (AMG-3)


1 256
257.................. 512
nn
byte1
nn
byte2
....
= actual value

This format is quite unusual and this is the reason for which is not possible to define the Amiga format under the standard classes. Usually we speak of a GCR encoded MFM.
Knowing this structure, we will be able to modify some elements of it, keeping the functionality of the format, but making it different enough from the standard to avoid the possibility of faithful reproduction from a copier.
The program then will have to find the non-standard information included in the format.

A good protection scheme should keep intact the main tracts of the Amiga format so that the disk appears to be standard if analyzed with a standard disk utility, it should allow one or more removable installations on the hard disk, without the need of a key disk, it should allow the serialisation of the product and it should advise the user in case of a failure during the protection check.

4. SILVER MASTER

Finally, after the Gold master and the format file are ready, the duplication firm will print five copies called "Silver masters". Two of those copies will be given to the developer for final approval, two will be kept for production and quality control and the last will be filed together with the Gold masters.
The testing of a Silver master, mainly if it includes protection, is a procedure that should recall beta testing. This notwithstanding, this control is often just a functionality test, while remaining the last chance to verify the work done. From my personal experience I can say that is not unusual to include the last fixes to a product in this phase.

5. DUPLICATION

The duplication process cannot be controlled directly by the developer, but is possible to ask to the duplication firm to agree to high quality standards to obtain a better product. It is a good idea to ask:

- a good magnetic medium
- a window margin not above 87.5% (using MDC specification, 75% on other systems)
- to precompensate the product

To say that the window margin should not exceed 75%, means to execute a verification procedure on the duplication system, in an area that is narrower than the original bitcell. Doing so the bit is validated only if it is readable in the central area of the bitcell thus guaranteeing that the disk will be readable even by a drive misaligned, in one way or the other, by 12.5%.
At the end we will have a smaller number of rejects from the users that do not have a perfectly aligned system, the most frequent ones. In our firm, we call this procedure as "duplication for the real world" and we think of it as necessary, even if it means a higher percentage of reject during quality control.

To precompensate the medium means to use a special procedure, written into the precompensation file and supplied by MDC, that on the basis of the encoding used, will analyze the disk and identify some special state transition that could create problems. In other word, the magnetic particles on the medium tend to attract or repel each other as all magnets. During precompensation, the particles that attract each other will be kept more distant and the ones that repel each other will be kept closer, so that the natural migration of those particles will bring our bit right in the middle of the bitcell. This procedure allows a narrower window margin and a longer life for the data on the disk.

Unfortunately, to verify these parameters you need several very expensive pieces of equipment, so that usually a developer has to trust the duplication firm he has chosen.

6. QUALITY CONTROL

Using these procedures, the quality control is performed at each step of production, so that the final product does not need special verifications. The only error that is possible in this system is due to a misalignment in the production drive, that is the same one that performs the verification. In this case both errors will compensate each other and the machine will wrongly think that the copy is OK.

This problem is quite rare, because any serious duplication firm checks often the alignment of their drives, but it still may happen.

For this reason a percentage between 5 and 20% of the production should be checked on the target system, according to the complexity of the format used.

7. FINAL NOTES

We should keep in mind that an increase in quality due to tighter quality control parameters or to complex copy protection schemes will reflect on the duplication cost of the finished product, that in any case should not exceed, for medium to large batches, the cost of a blank floppy disk as seen by the end user.

Following those rules we will obtain copies of our product that are, under all aspects, better than the original, and we will have taken an important step toward the realization of a good product, able to satisfy both the user and our financial expectations.

APPENDIX A - Amiga format written in Freeform language

```
NAME     =   ~Amiga DOS - 3.5" 2*11*512 80T 135tpi noMag (index)~
NUMBER   =   ~032-00044~
REV      =   ~06~
;
;format =    Double-sided. Tracks 0-79 have 11 512-bytes MFM
;            sectors/track. Formatted in 79 to 0 direction due
;            to torque and startup of 3.5" media.
;note   =    Disk is written synchronously with INDEX pulse.
;
;drive  =    double-side, any azimuth, 600 oersted 3.5" media,
;            135tpi @ 600rpm
DRV      =   "20234060"         ; Drive definition
PRECOMP  =   "mfm.1"            ; Precompensation filename
;
;            MEE S.p.A.
;            via Villoresi, 7
;            20143 Milano - Italy
;
;            (C) Magnetic Design Corporation
;
;*******************************************************************
;
;   =========================================
;       OUTER BLOCK OF FORMAT DEFINITION
;   =========================================
;
DEFINE FORMAT
    encoding = AMG-1
    bitcell = 1000
    setread              ; allow for non-index aligned read-in
        noindex
    setwrite             ; align write with INDEX
        index
    for I = 0 to 79 do REVERSE
END

DEFINE REVERSE
    for TRACK = (79-I) do BOTH-SIDES
END

DEFINE BOTH-SIDES
    for SIDE = 0 to 1 do MASTER-IMAGE
END
;
;   =========================================
;             MASTER TRACK IMAGES
;   =========================================
;
DEFINE MASTER-IMAGE
    encoding = AMG-1
    gap,count=150,byte=00
        for SECTOR = 0 do SECTOR-ZERO-IMAGE
        for SECTOR = 1 to 10 do SECTOR-IMAGE
    fill=00h
END
;
```

```
;   ========================================
;        SECTOR IMAGES FOR ALL SECTORS
;   ========================================
;
DEFINE SECTOR-ZERO-IMAGE
    FIND                ; finds first physical sector to start analysis
        gap,count=2,byte=00
        mark,count=2,byte=0A1h,clock=0Ah     ; MARK
        startedc="XOR16",EDC0=00             ; Init ID EDC
        encoding = AMG-2
            header,byte=0FFh                 ; format type 1.0
            header,byte=(TRACK * 2) + SIDE   ; Track #
            header,byte=SECTOR               ; Sector #
            header,byte=11-SECTOR            ; Offset
            data,count=16,byte=00            ; OS data
        edc,byte=EDC0D
        edc,byte=EDC0C
        edc,byte=EDC0B
        edc,byte=EDC0A
    ENDFIND
    startedc="XOR16",EDC0=00                 ; Init Data EDC
    encoding = AMG-3
    edc,byte=EDC0D
    edc,byte=EDC0C
    edc,byte=EDC0B
    edc,byte=EDC0A
        data,count=512,byte=0F6h
    encoding = AMG-1
END

DEFINE SECTOR-IMAGE
    gap,count=2,byte=00
    mark,count=2,byte=0A1h,clock=0Ah
    startedc="XOR16",EDC0=00
    encoding = AMG-2
        header,byte=0FFh
        header,byte=(TRACK * 2) + SIDE
        header,byte=SECTOR
        header,byte=11-SECTOR
        data,count=16,byte=00
    edc,byte=EDC0D
    edc,byte=EDC0C
    edc,byte=EDC0B
    edc,byte=EDC0A
    startedc="XOR16",EDC0=00
    encoding = AMG-3
    edc,byte=EDC0D
    edc,byte=EDC0C
    edc,byte=EDC0B
    edc,byte=EDC0A
        data,count=512,byte=0F6h
    encoding = AMG-1
END
```

CDTV

BOOKMARK and CARDMARK DEVICE DRIVER MANUAL

By Carl Sassenrath, Pantaray, Inc., Ukiah CA

Version: 1.0 – August 20, 1991

1. **Overview**
    - *Bookmark Memory*
    - *Device Driver*
    - *Cardmarks*
    - *Contents*

2. **Principles**
    - *Initialization*
    - *Identifiers*
    - *Aging*
    - *Program Access*

3. **Programming**
    - *Device Driver*
    - *Performing Commands*
    - *Aborting Commands*
    - *Commands*
    - *Creating a Bookmark*
    - *Writing and Reading*
    - *Deleting*
    - *Creating Cards*
    - *Testing Hints*

4. **Command Reference**

© 1991 Commodore Electronics, Ltd.
All Rights Reserved.

BOOKMARKS and CARDMARKS

CDTV Developer Manual

1. Overview

Bookmarks provide a means of storing CDTV application data across machine resets and power button shut-downs. For the vast majority of CDTV machines in the world (those not equipped with a floppy disk or hard disk drive), bookmarks serve as the only technique available to program designers for semi-permanent data storage.

Bookmarks were designed primarily to hold tiny hints of data that could be utilized by an application to return the user to a previously marked position within the program (hence the name "Bookmark"). For example: an encyclopedia may provide bookmarks to help users locate commonly referenced topics; a cookbook may indicate the previous recipe prepared; a math tutorial may recall the last lesson successfully completed; or a game may store its top five scores.

There is of course a limit to how many bookmarks can be stored on a CDTV. They should not be thought of nor used as a substitute for the larger, permanent storage medium of floppy disk.

Bookmark Memory

Every CDTV comes factory equipped with a small module of special memory for the storage of semi-permanent data. This memory is not located in the normal main memory spaces and is not configured into the operating system memory allocation lists. Instead, it has its own private location which is accessed only through the proper software protocols.

Bookmark memory is a "line-backed" RAM (Random Access Memory) not a battery backed RAM. It will persist and hold its data only while AC power is connected to the unit. The CDTV power switch will not affect the memory, but unplugging the unit or a power failure will destroy its contents. (The front panel clock display serves as a power indicator for the bookmark RAM.) In the current hardware, there is no provision for a capacitive power reserve to carry across momentary power failures or the relocation of the unit to some other room.

Device Driver The Bookmark Device Driver is responsible for managing all accesses to the bookmark memory. It resides in the Operating System ROM of every CDTV and operates in a standard fashion as an Amiga kernel level (Exec) device driver. The device driver was created with three purposes in mind:

- Managing the limited amount of memory available;
- Sharing of the memory between multiple applications executing concurrently or during different sessions;
- Removing old bookmarks in favor of new ones.
- Hiding the location and size of the memory so that it may be altered or expanded in future hardware designs.

The device driver organizes memory to appear as a "limitless" pool of independent bookmarks. It provides a sort of "file system" that allows multiple applications to store small data segments in a moderately simple way. The driver is not a standard file system however. During its design stage the decision was made not to use an Amiga file system (e.g. RAM disk) because it was more important to optimize the storage capacity available to applications rather than optimize the access mechanism. As a result, the storage overhead of a bookmark is low thus more bookmarks can be saved.

Cardmarks In addition to the factory equipped bookmark RAM, CDTV also provides an alternate storage mechanism: credit card style memory cards. These cards allow consumers to expand the memory capabilities of their units in a number of ways. The cards can be used for:

- Expanded bookmark storage (optionally battery backed)
- ROM based Operating System enhancements (libraries and devices)
- ROM based program applications
- Expansion RAM for increased main memory
- Recoverable RAM disks (optionally battery backed)
- Diagnostic testing
- Special hardware enhancements

The bookmark device driver has the capability to access these cards when expanded bookmark storage is desired. They can be formatted for use with bookmarks (often called cardmarks) and managed in a fashion identical to that used with bookmarks.

Applications have major advantages when it comes to storing data in cardmarks. The card's memory is normally much larger than the line-backed bookmark memory, which means a greater number of bookmarks can be saved, and each bookmark can be larger in size. Also, memory cards are usually backed up with battery power, thus protecting them from loss of power.

There is, however, one serious drawback to a memory cards: they can be removed from the machine. Just because your program wrote a cardmark does not mean you will always be able to access it. Also, a user may select an older memory card containing out of date cardmarks. If not dealt with properly in your application, this could lead to consumer confusion. In some cases you may want to create a date stamp in your data.

One possible approach for using cardmarks is to first check to see if a card is available. If so, use it, otherwise use a normal bookmark. It is a good idea to tell the user that a "bookmark is being placed" on the card, so that he can be aware that the card is being used and note what card it is.

2. Principles

This section describes the general principles of bookmarks and the bookmark device driver. These principles hold for both bookmarks and cardmarks created with the CDTV bookmark device.

Initialization

When CDTV is started for the first time after a full power loss, its line-backed bookmark memory is configured and initialized. The memory is scanned to determine its size, it is cleared to zero, and a header is placed within it, marking that it is free. The bookmark device driver is initialized and becomes available as a standard Exec device.

If a memory card is present, it is examined to determine whether it is already being used for bookmarks, free to be used for bookmarks, or not available for use. If the card is being used for bookmarks, its device driver is initialized. If the card is free, it is scanned to determine its size, cleared to zero, setup for bookmarks, and its driver is initialized. If a card is not available, it is left unused, and the cardmark driver is not initialized.

Identifiers

In order for multiple application to access their own specific bookmarks, each bookmark must contain an identifier to distinguish it from the others. Each bookmark is labelled with a unique code called a Bookmark Identifier or BID. A BID must be used to create, read, write, and delete a specific bookmark.

A BID is made up of two parts: a manufacturer code and a product number. The manufacturer code identifies the company or organization to which the bookmark belongs. These codes are issued by Commodore Electronics to anyone wishing to develop for CDTV. There are 64K codes available. We can only hope that we will someday run out. To obtain such a code you must call Commodore. Please do not make up your own codes – this can have a bad effect on consumers There are a lot of other more interesting challenges in the world.

The first few manufacturer codes are reserved for Commodore. For example, should you want to access the CDTV preferences, you will use a Commodore code of 0001. Other Commodore codes are used for testing, BID expansion, memory expansion, examples, etc.

The product number identifies the application related to the bookmark. It is normally just a product number created by you. There are 64K possibilities, so you are limited to making only 65,535 applications. At that point you can retire or start a new company and get a new manufacturer code. It is sometimes useful to divide the product number to allow multiple bookmarks for a single product. For example, the upper twelve bits may be what you use to identify your product and the lower four bits may indicate the bookmark number.

The actual form of a BID is specified in the *bookmark.h* file as:

```
structure BookmarkID
{
   UWORD Manufacturer;
   UWORD Product;
};
```

A macro called MAKEBID is also defined for creating BID constants.

Content The content of your bookmark may take any form needed by your application. You may want to use it for storing sector/page/screen/button/line numbers to indicate a return point, a hash value that determines where to resume or what you were doing, character strings, path/directory/file names, high-scores, etc.

The guiding rule is whatever you store, make it as small as possible. With limited space in the bookmark memory, the smaller the average bookmark, the more will fit. If you can get by with just a few bytes, do so. For example don't waste space by storing 32 bit numbers that only range from one to ten. Chop them down to at least a byte size.

Another way to reduce bookmark size is to use various index, offset, and table approaches. For example, if you need to save four text strings out of a set of 4000, you could save space by placing the strings in a file, and saving only the file byte offsets in the bookmark. Better yet, store an index into a table of offsets. If you need to store a set of file names, make your file names numeric and store them as binary in a bookmark. If you are writing a game that can be saved in one of 25000 states and you have 50 MB free on your CD, you could save the index in a bookmark, and use the CD to store the actual state information. Finally, if you must save textual words, save tokens. You might even want to use the approach of letting your most common 127 words be stored as bytes, and the remaining set stored as 16 bit words.

Bookmarks over a certain size may not be storable within bookmark memory. The device driver will not accept a bookmark that is larger than 1/16 the total size of the bookmark RAM. To determine the maximum size allowed, use the driver's MAXSIZE command.

Aging Over time it is likely that the CDTV consumer is going to fill his bookmark memory to the point where no more bookmarks will fit. When this happens, the device driver will start removing older bookmarks in order to make more space. To the consumer, it just looks like the machine forgot something done "a while back". This is a natural way for people to reason about it.

Two factors determine what bookmark will be removed: its priority and its age. The priority of a bookmark is assigned by the programmer when it is created, or it can be modified later. It can range from -128, a very low priority, to +127, very high. The default or standard value is zero. Bookmarks with a higher priority will remain over a longer period relative to other bookmarks.

You may have a difficult time deciding the priority for your bookmarks. In general, consider the consumer's interest and use the lowest priority possible. By application categories, the following values are suggested:

+40	*Education*
+20	*Reference*
0	*Arts and Leisure*
–20	*Music*
–40	*Entertainment*

What's the logic here? Just because Timmy plays 27 games in an evening, doesn't mean that he should blow away Dad's Car Repair Encyclopedia, Mom's Italian Language Lesson, or Grandma's Blueberry Pie Recipe (especially that). Think about who bought the machine.

Of course, every developer is naturally going to think that his or her application is more important than the others. That approach is not going to work. If everyone sets their priority to 60, the net effect is the same as everyone setting it to zero. When in doubt, use zero.

Another approach is to let the user decide. If you present a menu for saving bookmarks, you might allow the user to decide if the item is "very important" or "no so important". Don't give 256 priority choices, just a few to keep it simple.

As a bookmark remains in memory it ages. The age is an indication of how long it has been hanging around *unreferenced*. As a bookmark ages it becomes more likely that it will be removed to make room for newer bookmarks.

Whenever a bookmark is referenced through a read or a write command its age is reset. This causes a bookmark to be "reborn" giving it an extended life. This seems like a natural approach, as the most recently used applications should have the "youngest" bookmarks.

Program Access Access to bookmarks is achieved through the standard Exec Device Driver interface. The device driver is accessed by calling the Exec Library function **OpenDevice()**. For Bookmark RAM the name "bookmark.device" is used; for Cardmark RAM use "cardmark.device". The BID is passed in the Unit number parameter to **OpenDevice()**.

```
error = OpenDevice("bookmark.device, MY_BID, IOReq, 0);
```

This must be done before attempting to send any commands to a bookmark or the device driver, even when the bookmark has not yet been created. When a bookmark with the given BID does exist, the IOReq will be linked to it through a couple of its internal fields.

When accessing a bookmark in any way, it is a good idea to open the bookmark, make changes, and then close it. Do not leave it open for the full length of your application.

PROGRAMMERS BEWARE:

1. Bookmarks are garbage collected and relocated from time to time. Do not attempt to access a bookmark directly in memory, it may move on you.
2. Both the size and location of the Bookmark RAM will change in coming months as CDTV is improved. Your application will remain unaffected so long as it only accesses bookmarks through the specified device driver.

Once a bookmark has been opened, you can read it, over-write it, update it, clear it, etc. The details of these operations will be discussed in the programming chapter and in the reference chapter.

3. Programming

This section describes the details of programming to obtain the best results from the bookmark and cardmark devices. It will explain the primary approach for using the device driver, creating and accessing bookmarks, as well special information on initializing and formatting new memory cards.

Device Driver

The bookmark device is a standard Exec style device driver as documented in Chapter 19 of the *Amiga ROM Kernel Reference Manual: Libraries and Devices* (RKM). All device commands are performed through I/O requests sent to the device and all of these commands are executed synchronously (there are no asynchronous commands). The device also supports the formatting and initialization of other memory devices such as memory cards.

The bookmark driver resides in the Operating System ROM of every CDTV. It is referenced by programs through its device node name "**bookmark.device**" or "**cardmark.device**" depending on the type of storage desired. As is the accepted practice for Exec device drivers, all characters in the device names must be lower case.

The bookmark device driver operates with a standard IO request structure **IOStdReq** as found in the header file *exec/io.h*. You can locate this structure within your application's program data segment, or you can allocate it in either CHIP or FAST memory. It must be properly initialized before being used (See examples in the *ROM Kernel Manual* or below). You can, of course, reuse the same I/O request structure as much as you need.

The sequence to initialize a typical IO request might look like:

```
struct MsgPort *IOPort;
struct IOStdReq *IOReq;

IOPort = CreatePort(0,0);
if (IOPort == NULL) Error(NO_PORT);

IOReq = CreateStdIO(IOPort);
if (IOReq == NULL) Error(NO_REQUEST);
```

The **Error()** function and error constants are not part of the system, they are for you to provide. Normally they would free and close any resources, print a message to the user, and exit.

Once initialized you can open the device with the **OpenDevice()** function. This finds the device driver and binds it to the request:

```
if (OpenDevice("bookmark.device", MY_BID, IOReq, 0))
        Error(OPEN_DEV);

if (OpenDevice("cardmark.device", MY_BID, IOReq, 0))
        Error(OPEN_DEV);
```

As shown the unit number (second parameter) contains the Bookmark Identifier (BID). The flags (last parameter) should be zero for future compatibility.

If the device cannot be opened, an error is returned. This happens if the device driver for a particular type of memory cannot be found. The "bookmark.device" should always open (unless the memory is defective). The "cardmark.device" will only open if there is a valid memory card (see section on "Initializing Cards").

These requests are now ready to be put to use for I/O. The examples in the rest of this chapter will assume that the device has been opened as shown above. When finished with the requests, they should be passed to **CloseDevice()** before being deleted or freed.

Performing Commands

Bookmark device commands are performed in exactly the same fashion as other Exec devices. The **DoIO()** function will perform synchronous commands, that is, it will not return until the command as completed. The **SendIO()** function also works, but provides no extra advantage because of the synchronous nature of the device.

Before performing a command, the IO request structure must be setup with the various parameters of the command. For instance, a read might be setup:

```
IOReq->io_Command = CMD_READ;
IOReq->io_Offset = 0;
IOReq->io_Length = 12;
IOReq->io_Data = Buffer;
```

The **io_Command** field always indicates the command to execute. The command constants are supplied in the header file *bookmark.h*. The **io_Offset, io_Length,** and **io_Data** fields will vary depending on the command. Normally, the **io_Offset** contains the starting point for a command. In the case of the READ above, it holds the starting byte number. The **io_Length** contains either the size of the operation, or as in the case above, the terminating value. The **io_Data** is used in only a few commands when either a data buffer is needed or additional parameters are required. It should be set to NULL for all other commands.

To perform a synchronous command, pass the request to **DoIO()**:

```
DoIO(IOReq);
```

The function will return when the command has completed. If an error occurs in the command, it will be returned in the **io_Error** field of the request. A zero indicates no error. The error value is also returned from **DoIO()** so you may want to take advantage of it:

```
if (DoIO(IOReq)) Error(IO_FAILED);
```

The actual error value will depend on the command executed. See the header file *bookmark.h* for error values.

For several commands there is additional information in the **io_Actual** field of the request. This information is dependent on the command executed. Normally it is related to the value placed in the **io_Length** field of the request. For example, a READ would return the total number of bytes read.

Aborting Commands Commands cannot be aborted because the bookmark device driver only processes them in a synchronous fashion.

Commands All of the bookmark device driver commands are defined in the *bookmark.h* header file. From these commands the standard set used by CDTV applications are:

Command	Num	Description
CMD_READ	2	Read data from bookmark
CMD_WRITE	3	Write data to bookmark
CMD_UPDATE	4	Reset bookmark age
CMD_CLEAR	5	Clear bookmark contents
BD_CREATE	13	Create new bookmark
BD_DELETE	14	Delete bookmark
BD_MAXSIZE	15	Return maximum bookmark size
BD_SIZEOF	17	Return the size of a bookmark
BD_SETPRI	18	Set the priority of a bookmark

The remaining commands are designed for special purposes and would not normally be a part of applications.

Some of these commands are helpful for debugging and testing your applications, and should be removed before your final release. A few other commands are meant to be used by memory card vendors as a means of formatting new memory cards. Specialized OEMs for vertical applications may also use these commands.

Command	Num	Description
CMD_RESET	1	Reset the bookmark memory
BD_TYPEMEM	9	Return RAM type
BD_SIZEMEM	10	Return RAM size (destructive)
BD_INITMEM	11	Initialize RAM
BD_CREATEDEV	12	Create new device driver
BD_AVAIL	16	Return number of bytes free in memory
BD_CHECK	19	Checksum bookmark memory
BD_PURGE	20	Purge bookmark memory
BD_DUMP	21	Dump bookmark memory to main memory
BD_LOAD	22	Load bookmark memory from main memory

Creating a Bookmark

Creating a new bookmark is easy and is absolutely necessary before the bookmark can be used for writing, reading, or other operations.

To create a bookmark simply follow these steps:

1. Call Commodore to obtain a manufacturer id number. You must do this if you want to use bookmarks in your applications, and Commodore has agreed to give them out without a fuss. So for this example, if your number is 0000, you might create a bookmark identifier of:

```
#define MY_BID 0x00000001
```

2. Determine what you are going to store. Use a data structure if you so desire:

```
struct MyBookmark
{
    UWORD SectorNum;
    UBYTE PathNum;
    UBYTE UserOptions;
};
```

3. Calculate the maximum size of your bookmark. If it is a structure you should be able to get by with "sizeof" in most cases. If you have variable length bookmarks, you will need to know the maximum length.

```
#define MAX_BOOKMARK sizeof(struct MyBookmark)
```

4. Create the I/O reply port and the I/O request:

```
struct MsgPort *IOPort;
struct IOStdReq *IOReq;

IOPort = CreatePort(0,0);
if (IOPort == NULL) Error(NO_PORT);

IOReq = CreateStdIO(IOPort);
if (IOReq == NULL) Error(NO_REQUEST);
```

5. Open the bookmark device driver. Use your own BID even though it should not yet exist:

```
if (OpenDevice("bookmark.device", MY_BID, IOReq, 0))
        Error(OPEN_DEV);
```

6. Determine and set the aging priority of your bookmark.

```
IOReq->io_Node.ln_Pri = -10;
```

7. Setup the parameters to the create command:

```
IOReq->io_Command = BD_CREATE;
IOReq->io_Offset  = MY_BID;
IOReq->io_Length  = MAX_BOOKMARK;
IOReq->io_Data    = 0;
```

8. Now you are ready for action. Create the bookmark:

```
DoIO(IOReq);
```

9. Do not assume that it worked. Check for errors:

```
switch (IOReq->io_Error)
{
case 0:
   msg = NULL;
   break;
case BDERR_TOOBIG:
   msg = "bookmark is larger than allowed";
   break;
case BDERR_EXISTS:
   msg = "bookmark with this identifier already exists";
```

```
      break;
   case BDERR_NOSPACE:
      msg = "no space is available for bookmark"
      break;
   }

   if (msg)
   {
      printf("ERROR IN CREATE: %s\n", msg);
      Handle error
   }
```

That's it. If there were no errors you know have a bookmark. The command actually allocated the specified amount of space within the bookmark RAM, pushing older bookmarks out if necessary.

Note: Be prepared for a short delay (milliseconds) from time to time. The bookmark memory space may need to be compacted if one or more of the older bookmarks must be deleted. This operation is performed during the create command. The length of the delay is variable depending on the number, size and positions of bookmarks. Also larger card memories will have longer delays.

It is important to realize that even though a BID is used in the call to **OpenDevice()**, there will not be bookmark memory associated with the request until the create has been done.

Writing and Reading Once a bookmark exists, you can use the standard Exec I/O commands for writing its contents and reading it back.

To write a bookmark:

1. Use the same BID and data structure you developed earlier in the create operation.

2. Create the I/O reply port and the I/O request:

```
   struct MsgPort *IOPort;
   struct IOStdReq *IOReq;

   IOPort = CreatePort(0,0);
   if (IOPort == NULL) Error(NO_PORT);

   IOReq = CreateStdIO(IOPort);
   if (IOReq == NULL) Error(NO_REQUEST);
```

3. Open the bookmark device driver. Use your BID here:

```
if (OpenDevice("bookmark.device", MY_BID, IOReq, 0))
        Error(OPEN_DEV);
```

4. Put data into your bookmark structure:

```
struct MyBookmark MB;

MB.SectorNum = Sector;
MB.PathNum = Path;
MB.UserOptions = Options;
```

5. Setup the parameters to the write command:

```
IOReq->io_Command = CMD_WRITE;
IOReq->io_Offset  = 0;
IOReq->io_Length  = sizeof(MB);
IOReq->io_Data    = &MB;
```

6. Write the bookmark:

```
DoIO(IOReq);
```

7. Check for errors:

```
switch (IOReq->io_Error)
{
case 0:
   msg = NULL;
   break;
case BDERR_NOMARK:
   msg = "bookmark does not exist";
   break;
case BDERR_PASTEND:
   msg = "attempt to write more than allowed";
   break;
case BDERR_BADARG:
   msg = "bad I/O argument was passed"
   break;
}

if (msg)
{
   printf("ERROR IN WRITE: %s\n", msg);
   Handle error
}
else
   printf("%d BYTES WRITTEN\n", IOReq->io_Actual);
```

If the **io_Offset** has a value other than zero, it is used as a byte offset into the bookmark. This allows you to change subfields of the bookmark independently.

After the command, the **io_Actual** field indicates the number of bytes written.

To read a bookmark, use the same steps 1, 2, and 3, skip step 4, then:

5. Setup the read command:

```
IOReq->io_Command = CMD_READ;
IOReq->io_Offset  = 0;
IOReq->io_Length  = sizeof(MB);
IOReq->io_Data    = &MB;
```

6. Read the bookmark:

```
DoIO(IOReq);
```

7. Check for errors. Use the same approach as above.

Deleting

If your bookmark is no longer needed nor desired, remove it from memory to make room for other applications.

To delete a bookmark:

1. Use the same BID and data structure you developed earlier in the create operation.

2. Create the I/O reply port, the I/O request, and open the device as shown in the previous examples.

5. Setup the parameters to the delete command:

```
IOReq->io_Command = BD_DELETE;
IOReq->io_Offset  = 0;
IOReq->io_Length  = 0;
IOReq->io_Data    = 0;
```

6. Delete the bookmark:

```
DoIO(IOReq);
```

7. Check for errors:

```
switch (IOReq->io_Error)
{
case 0:
   msg = NULL;
   break;
case BDERR_NOMARK:
      msg = "bookmark does not exist";
      break;
   }

   if (msg)
   {
      printf("ERROR IN DELETE: %s\n", msg);
      Handle error
   }
```

Creating Cards

Card memory vendors or publishers may in some cases need to devise a means for initializing new memory cards and prepare them for either bookmark or non-bookmark use.

It should be stressed that the code in this section is not to be used in CDTV applications. If you do, expect your program to break in a few months when the hardware is revised.

Memory cards are classified into the following types. The first column is the value (hex or ASCII) present in the first word of the memory.

`'BK'`	`bookmark initialized RAM`
`'RW'`	`expansion RAM`
`'RD'`	`recoverable RAM disk`
`'RO'`	`special system/application ROM`
`'OK'`	`reserved for something else`
`$1111`	`system diagnostics`

When the system bootstraps, the bookmark device examines this memory to determine its type. If it is 'BK' then it assumes the memory contains valid bookmarks. If it is any of the other types, it ignores the card, otherwise it initializes the memory for new bookmarks.

If you are selling a card to be used as expansion RAM or a special RAM disk you need to pre-initialize your card to 'RW' or 'RD. There is another way to do this safely from a utility CD or floppy, but it is beyond the scope of this document.

To create a memory card with pre-initialized bookmarks takes a few special commands. The correct sequence is:

1. Define the location of the card memory and its maximum size. **Do not use this in CDTV applications!**

```
#define CARD_MEM 0xE00000
#define MAX_SIZE 0x080000
```

2. Create the I/O reply port and the I/O request:

```
struct MsgPort *IOPort;
struct IOStdReq *IOReq;

IOPort = CreatePort(0,0);
if (IOPort == NULL) Error(NO_PORT);

IOReq = CreateStdIO(IOPort);
if (IOReq == NULL) Error(NO_REQUEST);
```

3. Open the bookmark device driver using a zero BID:

```
if (OpenDevice("bookmark.device", 0, IOReq, 0))
        Error(OPEN_DEV);
```

4. Check the type of memory:

```
IOReq->io_Command = BD_TYPEMEM;
IOReq->io_Offset  = 0;
IOReq->io_Length  = 0;
IOReq->io_Data    = CARD_MEM;
DoIO(IOReq);
if (IOReq->io_Actual) /* Is it available or not? */
{
   puts("Card memory already in use");
   Handle error;
}
```

5. Find the size of the memory. This is a destructive test so use it with care.

```
IOReq->io_Command = BD_SIZEMEM;
IOReq->io_Offset  = 0;
IOReq->io_Length  = MAX_SIZE;
IOReq->io_Data    = CARD_MEM;
DoIO(IOReq);
if (IOReq->io_Actual == 0) /* Is it there? */
{
   puts("Memory card is not present");
   Handle error;
}
```

6. Initialize the memory, getting it ready for bookmarks:

```
IOReq->io_Command = BD_INITMEM;
IOReq->io_Offset  = 0;
IOReq->io_Length  = IOReq->io_Actual; /* the size */
IOReq->io_Data    = CARD_MEM;
DoIO(IOReq);
```

7. At this point you can reset your machine or initialize the cardmark device driver. In theory it would probably be better to reset your machine, but if this creates problems for your initializing program you can start the cardmark device driver with the code below.

```
/* Allocate device name in memory */
name = AllocMem(strlen("cardmark.device")+1,0);
if (!name) HandleError(...);  /* don't return */
strcpy(name, "cardmark.device");

IOReq->io_Command = BD_CREATEDEV;
IOReq->io_Offset  = 0;
IOReq->io_Length  = name;
IOReq->io_Data    = CARD_MEM;
DoIO(IOReq);
if (!IOReq->io_Actual)
   HandleError("cannot initialize cardmark device");
```

The astute programmer may recognize that it is possible to create bookmark devices anywhere in memory. All you need to do is perform an **AllocMem()** then pass the memory address on to **BD_INITMEM**, then **BD_CREATEDEV**. This is one way to simulate the operation of bookmarks without actually accessing the bookmark or cardmark memories. Again, this should only be done for testing purposes and not applications.

Testing Hints

Finally, if your application makes extensive use of bookmarks, you may want to use a few special commands to make your testing easier.

The bookmark and cardmark memories as a whole or in part can be transferred to and from main memory. This allows you to create programs that load an entire set of bookmarks before a test begins. See the reference section for details on the **BD_DUMP** and **BD_LOAD** commands.

Also, it is a wise idea to verify before application release that your bookmark memory is free from wild pointer hits and corruption. To do this, do the **BD_CHECK** command at the start of your program, then again every so often. Set the **io_Offset** field TRUE when you first do the command and after every bookmark command that modifies the memory. Whenever you receive a non-zero result, something has modified bookmark memory! See the reference section for more information.

Bookmark Device Driver Command Reference

1 CMD_RESET

bookmark.device

Reset the bookmark/cardmark memory to its initial, cleared power-on state. (For testing purposes only.)

COMMAND | FUNCTION | INTERNAL | INTERRUPT

Inputs

```
io_Command= CMD_RESET
io_Offset = 0
io_Length = 0
io_Data   = 0
```

Outputs

```
ErrorCode = io_Error
```

Description

This command resets bookmark or cardmark memory to its initial configuration state. The entire memory is cleared and all marks are lost. This operation is not suggested for released programs, but can be used for application testing when it is necessary to clear mark memory to a known state.

Note that for this command to work correctly, the device memory header must be intact. This command is not to be used to initialze new, blank, or unformated memory devices (see BD_INITMEM).

Example

```
extern struct IOStdReq *IOReq;             /* IO request opened earlier */

IOReq->io_Command = CMD_RESET;             /* That's all folks */
IOReq->io_Offset = 0;
IOReq->io_Length = 0;
IOReq->io_Data = 0;
DoIO(IOReq);
```

Errors

Related

BD_INITMEM
BD_CREATEDEV

CMD_READ

Read data from an existing bookmark.

COMMAND FUNCTION
INTERNAL INTERRUPT

Inputs

```
io_Command= CMD_READ
io_Offset = ByteOffset
io_Length = NumberOfBytes
io_Data   = Buffer
```

Outputs

```
ErrorCode = io_Error
BytesRead = io_Actual
```

Description

Transfer the specified number of bytes from the bookmark into memory. The bookmark must already exist (BD_CREATE) and contain data (CMD_WRITE).

If a -1 is specified for the length, the entire remaining contents of the bookmark are transferred.

The actual number of bytes read is returned in io_Actual.

If io_Offset + io_Length exceeds the size of the bookmark, a BDERR_PASTEND occurs and no data is read.

Each time a bookmark is read, its age is reset, extending the life of the bookmark in memory.

WARNING: Never attempt to access a bookmark directly in bookmark/cardmark memory space. Bookmark space is compacted from time to time and bookmarks may be relocated without your knowledge.

Example

```
extern struct IOStdReq *IOReq;           /* IO request opened earlier */

char Buffer[52];

IOReq->io_Command = CMD_READ;
IOReq->io_Offset = 0;
IOReq->io_Length = 52;
IOReq->io_Data = Buffer;
if (DoIO(IOReq)) ProcessError(IOReq);
printf("%d bytes read\n", IOReq->io_Actual);

IOReq->io_Command = CMD_READ;
IOReq->io_Offset = 10;
IOReq->io_Length = -1;
IOReq->io_Data = Buffer;
if (DoIO(IOReq)) ProcessError(IOReq);
printf("%d bytes read\n", IOReq->io_Actual);
```

Errors

NOMARK – no bookmark exists
PASTEND – attempt read more than possible
BADARG – bad I/O argument

Related

CMD_WRITE
BD_CREATE

3 CMD_WRITE

Write data to an existing bookmark.

COMMAND | FUNCTION | INTERNAL | INTERRUPT

Inputs

```
io_Command= CMD_WRITE
io_Offset = ByteOffset
io_Length = NumberOfBytes
io_Data   = Buffer
```

Outputs

```
ErrorCode = io_Error
BytesRead = io_Actual
```

Description

Transfer the specified number of bytes from memory into the bookmark. The bookmark must already exist or have been created with BD_CREATE.

The maximum size of the transfer is limited to the maximum size of the bookmark as specified in the original create operation. If a -1 is specified for the length, the full bookmark size is used for the transfer.

The actual number of bytes written is returned in io_Actual.

If io_Offset + io_Length exceeds the size of the bookmark, a BDERR_PASTEND occurs and no data is written.

Each time a bookmark is written, its age is reset, extending the life of the bookmark in memory.

WARNING: Never attempt to access a bookmark directly in bookmark/cardmark memory space. Bookmark space is compacted from time to time and bookmarks may be relocated without your knowledge.

Example

```
extern struct IOStdReq *IOReq;          /* IO request opened earlier */

char Buffer[] = "Save this string";

IOReq->io_Command = CMD_WRITE;
IOReq->io_Offset = 0;
IOReq->io_Length = strlen(Buffer) + 1;  /* string length with terminator */
IOReq->io_Data = Buffer;
if (DoIO(IOReq)) ProcessError(IOReq);
printf("%d bytes written\n", IOReq->io_Actual);

IOReq->io_Command = CMD_WRITE;
IOReq->io_Offset = 5;
IOReq->io_Length = 4;
IOReq->io_Data = &Buffer[5];            /* write just "this" */
if (DoIO(IOReq)) ProcessError(IOReq);
printf("%d bytes written\n", IOReq->io_Actual);
```

Errors

NOMARK – no bookmark exists
PASTEND – attempt write more than possible
BADARG – bad I/O argument

Related

CMD_READ
BD_CREATE

CMD_UPDATE

COMMAND FUNCTION
INTERNAL INTERRUPT

Reset the age of a bookmark.

Inputs

```
io_Command= CMD_UPDATE
io_Offset = 0
io_Length = 0
io_Data   = 0
```

Outputs

```
ErrorCode = io_Error
```

Description

Reset the age of a bookmark, thus extending its life in memory.

The age of a bookmark is used to determine its priority for being replaced by other bookmarks requiring memory. When a bookmark reaches a certain age, it becomes a candidate for removal so its memory can be reused. Whenever its age is reset, it is renewed for another full lifespan.

The aging priority of a bookmark is initially set by the BD_CREATE command, and it may be changed with the BD_SETPRI command.

The CMD_READ and CMD_WRITE commands automatically reset the age of a bookmark.

Example

```
extern struct IOStdReq *IOReq;                /* IO request opened earlier */

IOReq->io_Command = CMD_UPDATE;
IOReq->io_Offset = 0;
IOReq->io_Length = 0;
IOReq->io_Data = 0;
if (DoIO(IOReq)) ProcessError(IOReq);
printf("Updated\n");
```

Errors

NOMARK – no bookmark exists

Related

BD_CREATE
BD_SETPRI

5 CMD_CLEAR

bookmark.device

Clear the contents of a bookmark.

COMMAND FUNCTION INTERNAL INTERRUPT

Inputs

```
io_Command= CMD_CLEAR
io_Offset = 0
io_Length = 0
io_Data   = 0
```

Outputs

```
ErrorCode = io_Error
```

Description

Clear the entire contents of a bookmark to zero, but do not delete the bookmark.

This command is only needed if you wish to erase the contents of a bookmark, but not free its memory.

When a bookmark is created (BD_CREATE) it is cleared automatically.

The age of the bookmark is reset.

Example

```
extern struct IOStdReq *IOReq;              /* IO request opened earlier */

IOReq->io_Command = CMD_CLEAR;
IOReq->io_Offset = 0;
IOReq->io_Length = 0;
IOReq->io_Data = 0;
if (DoIO(IOReq)) ProcessError(IOReq);
printf("Cleared\n");
```

Errors

NOMARK – no bookmark exists

Related

BD_WRITE

9 BD_TYPEMEM

Return the type code of a particular module of the bookmark/cardmark memory space. (Not for Applications!)

COMMAND FUNCTION
INTERNAL INTERRUPT

Inputs

```
io_Command= BD_TYPEMEM
io_Offset = 0
io_Length = 0
io_Data   = MemoryRegion
```

Outputs

```
TypeCode = io_Actual
```

Description

This is an internal command used primarily by memory card manufacturers, vertical market developers, and snoopy application developers. It is used to help configure blank memory cards.

The code returned in io_Actual determines the current use and state of the memory. The values are:

>0 when the memory is already in use for bookmarks/cardmarks

=0 when the memory is available for use

<0 when the memory is not available (being used for diagnostics, expansion RAM, ROM, RAMDisk, etc.)

This command has no relationship or connection with memory controlled by the Exec library. It should not be used with general system memory.

Example

```
extern struct IOStdReq *IOReq;              /* IO request opened earlier */

IOReq->io_Command = BD_TYPEMEM;
IOReq->io_Offset = 0;
IOReq->io_Length = 0;
IOReq->io_Data = MemoryRegion;
DoIO(IOReq);
if (IOReq->io_Actual > 0) printf("Bookmark/Cardmark memory\n");
else if (IOReq->io_Actual == 0) printf("Available memory\n");
else printf("In use by something else\n");
```

Errors

Related

BD_SIZEMEM
BD_INITMEM
BD_CREATEDEV

10 BD_SIZEMEM

bookmark.device

COMMAND | FUNCTION | INTERNAL | INTERRUPT

Determine the size of a particular module of bookmark/cardmark memory. (Not for Applications!)

Inputs

```
io_Command= BD_SIZEMEM
io_Offset = 0
io_Length = MaxSize
io_Data   = MemoryRegion
```

Outputs

```
ByteSize = io_Actual
```

Description

This is an internal command which attempts to determine the size of a region of memory available for bookmarks. This command is normally executed after BD_TYPEMEM indicates that the memory is available for use.

This is a destructive memory test. It will alter the contents of the memory region. It should not be performed on active bookmark/cardmark or system memory.

Memory will be scanned to a resolution of 2K. The scan will be stopped when:

- The memory cannot hold a test value.
- The memory wraps over itself.
- The maximum size is reached as specified in io_Length .

Example

```
extern struct IOStdReq *IOReq;              /* IO request opened earlier */

IOReq->io_Command = BD_SIZEMEM;
IOReq->io_Offset = 0;
IOReq->io_Length = 256 * 1024;
IOReq->io_Data = MemoryRegion;
DoIO(IOReq);
printf("%d bytes in size\n", IOReq->io_Actual);
```

Errors

Related

BD_TYPEMEM
BD_INITMEM
BD_CREATEDEV

11 BD_INITMEM

bookmark.device

Initialize a particular region of bookmark/cardmark memory.
(Not for Applications!)

COMMAND | FUNCTION | INTERNAL | INTERRUPT

Inputs

```
io_Command= BD_INITMEM
io_Offset = 0
io_Length = MemorySize
io_Data   = MemoryRegion
```

Outputs

Description

This is an internal command which will initialize a region of memory for use as bookmarks/cardmarks. Its primary purpose is to "format" new memory cards. It clears the memory, sets up the memory header, and determines the maximum size of bookmarks for the memory.

This command is normally executed after BD_TYPEMEM and BD_SIZEMEM have been performed.

The io_Length parameter limits the amount of space to be used for storage of bookmarks (and the header). This allows you to reserve space in the memory region for other uses.

Example

```
extern struct IOStdReq *IOReq;             /* IO request opened earlier */

IOReq->io_Command = BD_INITMEM;
IOReq->io_Offset = 0;
IOReq->io_Length = MemorySize;
IOReq->io_Data = MemoryRegion;
DoIO(IOReq);
```

Errors

Related

BD_TYPEMEM
BD_SIZEMEM
BD_CREATEDEV

12 BD_CREATEDEV

bookmark.device

COMMAND FUNCTION
INTERNAL INTERRUPT

**Put a new bookmark/cardmark device on-line.
(Not for Applications!)**

Inputs

```
io_Command= BD_CREATEDEV
io_Offset = 0
io_Length = DeviceName
io_Data   = MemoryRegion
```

Outputs

```
DevBase = io_Actual
```

Description

This is an internal command used to create a new bookmark/cardmark device and put it on-line. It can be used to configure and install new memory cards. It will create and initialize an Exec device node and its function pointers, then add the device to Exec so that it may be accessed with OpenDevice().

The device name is passed as an argument. For cardmark devices, this name should be "cardmark.device". Be sure to allocate this name someplace where it will not be freed when your program exits.

This command does not initialize the memory region being installed. You must use BD_INITMEM before executing this command.

The device base address of the functioning device is returned in io_Actual. If a NULL is returned, the MakeLibrary() function failed (out of memory).

Example

```
extern struct IOStdReq *IOReq;          /* IO request opened earlier */
#define MARKNAME "cardmark.device"
char *name;

IOReq->io_Command = BD_SIZEMEM; IOReq->io_Offset = 0;
IOReq->io_Length = 256 * 1024; IOReq->io_Data = MemoryRegion;
DoIO(IOReq);
size = IOReq->io_Actual;

IOReq->io_Command = BD_INITMEM; IOReq->io_Offset = 0;
IOReq->io_Length = size; IOReq->io_Data = MemoryRegion;
DoIO(IOReq);

name = AllocMem(strlen(MARKNAME)+1,0);
if (!name) BadNews(); else strcpy(name,MARKNAME);

IOReq->io_Command = BD_CREATEDEV; IOReq->io_Offset = 0;
IOReq->io_Length = (LONG) name; IOReq->io_Data = MemoryRegion;
DoIO(IOReq);
if (!IOReq->io_Actual) BadNews();
```

Errors

Related

BD_TYPEMEM
BD_SIZEMEM
BD_INITMEM

13 BD_CREATE

bookmark.device

COMMAND FUNCTION INTERNAL INTERRUPT

Create a new bookmark/cardmark entry.

Inputs

```
io_Command= BD_CREATE
io_Offset = BookmarkId
io_Length = MaxBytes
io_Data   = 0 and ln_Pri = Pri
```

Outputs

```
ErrorCode = io_Error
```

Description

This command is used to create new bookmarks. This must be done before a CMD_WRITE command. A bookmark of the requested size will be allocated from the memory associated with the device you opened in the call to OpenDevice(). The io_Offset field must contain a valid bookmark identifier consisting of both a manufacturer id and product code. If the bookmark already exists you will receive an error, and the command will not be performed.

The maximum size of a bookmark is restricted to allow many bookmarks to share memory.

The priority of a bookmark comes from the ln_Pri field of the I/O Request node. This value establishes the initial age of the bookmark. It may range between -128 and +127. You will need to determine the importance of your bookmark relative to other bookmarks (see previous chapter).

The contents of a new bookmark are cleared to zero.

Example

```
extern struct IOStdReq *IOReq;              /* IO request opened earlier */

char Buffer[] = "new bookmark";

IOReq->io_Command = BD_CREATE;
IOReq->io_Offset = MY_BID;                  /* Must be valid Manuf/Prod Id */
IOReq->io_Length = 64;
IOReq->io_Data = 0;
IOReq->io_Node.ln_Pri = 0;
if (DoIO(IOReq)) ProcessError(IOReq);

IOReq->io_Command = CMD_WRITE;
IOReq->io_Offset = 0;
IOReq->io_Length = strlen(Buffer) + 1;      /* string length with terminator */
IOReq->io_Data = Buffer;
if (DoIO(IOReq)) ProcessError(IOReq);
```

Errors

TOOBIG – size is bigger than allowed
EXISTS – bookmark already exists with this id
NOSPACE – out of bookmark memory

Related

BD_DELETE
CMD_UPDATE
CMD_WRITE

14 BD_DELETE

bookmark.device

Delete a bookmark/cardmark entry.

COMMAND | **FUNCTION** | **INTERNAL** | INTERRUPT

Inputs

```
io_Command= BD_DELETE
io_Offset  = 0
io_Length  = 0
io_Data    = 0
```

Outputs

```
ErrorCode = io_Error
```

Description

This command deletes a bookmark and frees its memory.

Attempting to delete a non-existent bookmark will return an error.

If you attempt to access this bookmark once it has been deleted you will receive an error.

Example

```
extern struct IOStdReq *IOReq;              /* IO request opened earlier */

IOReq->io_Command = BD_DELETE;
IOReq->io_Offset = 0;
IOReq->io_Length = 0;
IOReq->io_Data = 0;
if (DoIO(IOReq)) ProcessError(IOReq);
```

Errors

NOMARK – no bookmark exists

Related

BD_CREATE

15 BD_MAXSIZE

Determine the maximum size allowed for a bookmark/cardmark.

COMMAND | FUNCTION | INTERNAL | INTERRUPT

Inputs

```
io_Command= BD_MAXSIZE
io_Offset = 0
io_Length = 0
io_Data   = 0
```

Outputs

```
ByteSize = io_Actual
```

Description

This command returns the maximum size permited for a bookmark. The value returned is dependent on the device you specified in the call to OpenDevice(). For example, the maximum size for cardmarks is normally larger than that of bookmarks due to the larger memory size. Also, memory cards may vary in size.

See the previous chapter for a discussion of the methods for reducing the size of bookmarks.

Example

```
extern struct IOStdReq *IOReq;            /* IO request opened earlier */

IOReq->io_Command = BD_MAXSIZE;
IOReq->io_Offset = 0;
IOReq->io_Length = 0;
IOReq->io_Data = 0;
DoIO(IOReq);
printf("%d byte maximum\n", IOReq->io_Actual);
```

Errors

Related

BD_CREATE
BD_AVAIL

16 BD_AVAIL

bookmark.device

Return the amount of free space available in bookmark / cardmark memory.

COMMAND FUNCTION INTERNAL INTERRUPT

Inputs

```
io_Command= BD_AVAIL
io_Offset = 0
io_Length = 0
io_Data   = 0
```

Outputs

```
ByteSize = io_Actual
```

Description

This command returns the total number of bytes available in bookmark or cardmark memory. The value returned is dependent on the device you specified in the call to OpenDevice().

Although a considerable amount of space may be available, applications are limited to a maximum bookmark size to allow room for other applicaitons.

Example

```
extern struct IOStdReq *IOReq;           /* IO request opened earlier */

IOReq->io_Command = BD_AVAIL;
IOReq->io_Offset = 0;
IOReq->io_Length = 0;
IOReq->io_Data = 0;
DoIO(IOReq);
printf("%d bytes available\n", IOReq->io_Actual);
```

Errors

Related

BD_MAXSIZE

17 BD_SIZEOF

Return the size of a bookmark/cardmark.

COMMAND FUNCTION INTERNAL INTERRUPT

Inputs

```
io_Command= BD_SIZEOF
io_Offset = 0
io_Length = 0
io_Data   = 0
```

Outputs

```
ErrorCode = io_Error
ByteSize = io_Actual
```

Description

This command returns the number of bytes allocated to a bookmark for data storage. It does not include system structures associated with the bookmark. The value is the same as that specified in the originating BD_CREATE.

The size of a bookmark is not affected by reading, writing, or clearing it. The size cannot be changed without deleting and recreating a new bookmark.

If no bookmark exists, this command returns an error.

Example

```
extern struct IOStdReq *IOReq;            /* IO request opened earlier */

IOReq->io_Command = BD_SIZEOF;
IOReq->io_Offset = 0;
IOReq->io_Length = 0;
IOReq->io_Data = 0;
if (!DoIO(IOReq)) ProcessError(IOReq);
printf("%d bytes in size\n", IOReq->io_Actual);
```

Errors

NOMARK – no bookmark exists

Related

BD_CREATE
BD_MAXSIZE

18 BD_SETPRI

bookmark.device

Change the aging priority of a bookmark.

COMMAND | FUNCTION | INTERNAL | INTERRUPT

Inputs

```
io_Command= BD_SETPRI
io_Offset = NewPri
io_Length = 0
io_Data   = 0
```

Outputs

```
ErrorCode = io_Error
OldPri = io_Actual
```

Description

This command changes the priority of a bookmark and returns the previous value. The higher the priority of a bookmark, the longer it will remain when memory has been exhausted and garbage collection has started.

Priorities can range between -128 and +127. The normal value is zero. Use zero if you are not sure.

The age of the bookmark is reset to a new value depending upon the new priority. (See also: CMD_UPDATE)

Before changing the priority of a bookmark, determine its importance relative to other bookmarks (see previous chapter).

The initial priority of a bookmark comes from the ln_Pri field of the I/O request node used for BD_CREATE. Its value establishes the initial age of the bookmark.

Example

```
extern struct IOStdReq *IOReq;           /* IO request opened earlier */

IOReq->io_Command = BD_SETPRI;
IOReq->io_Offset = -10;
IOReq->io_Length = 0;
IOReq->io_Data = 0;
if (!DoIO(IOReq)) ProcessError(IOReq);
printf("Old priority was %d\n", IOReq->io_Actual);
```

Errors

NOMARK – no bookmark exists

Related

BD_CREATE
BD_UPDATE

19 BD_CHECK

bookmark.device

Calculate the checksum for the entire bookmark/cardmark memory space.

COMMAND FUNCTION INTERNAL INTERRUPT

Inputs

```
io_Command= BD_CHECK
io_Offset = SetFlag
io_Length = 0
io_Data   = 0
```

Outputs

```
ErrorCode = io_Error
Checksum = io_Actual
```

Description

This command calculates the checksum of all words within the bookmark/cardmark memory space. It will return zero if nothing in memory has been modified since the last checksum command, otherwise it will return a new checksum value.

The new checksum result will not be updated internally unless the io_Offset is set TRUE (non-zero).

The primary purpose of this command is to support application debugging efforts. This command can be called at various points in a program to determine if the bookmark/cardmark memory is being corrupted by bad indirection pointers.

Example

```
extern struct IOStdReq *IOReq;              /* IO request opened earlier */

IOReq->io_Command = BD_CHECK;
IOReq->io_Offset = TRUE;                    /* Save checksum internally */
IOReq->io_Length = 0;
IOReq->io_Data = 0;
DoIO(IOReq);
printf("Checksum: %d\n", IOReq->io_Actual);

IOReq->io_Command = BD_CHECK;
IOReq->io_Offset = FALSE;
IOReq->io_Length = 0;
IOReq->io_Data = 0;
DoIO(IOReq);
if (IOReq->io_Actual)
     printf("Checksum different: %d\n", IOReq->io_Actual);
```

Errors

Related

20 BD_PURGE

bookmark.device

COMMAND FUNCTION
INTERNAL INTERRUPT

Purge the entire bookmark/cardmark memory.
(Not for Applications!)

Inputs

```
io_Command= BD_PURGE
io_Offset = 0
io_Length = 0
io_Data   = 0
```

Outputs

Description

This command erases the entire bookmark memory including all system headers. It is supplied for testing purposes (for use with BD_DUMP, BD_LOAD) and for special memory card vendors. **This command should not be used in normal applications.**

Example

```
extern struct IOStdReq *IOReq;              /* IO request opened earlier */

IOReq->io_Command = BD_PURGE;
IOReq->io_Offset = 0;
IOReq->io_Length = 0;
IOReq->io_Data = 0;
DoIO(IOReq);
printf("Checksum: %d\n", IOReq->io_Actual);
```

Errors

Related

BD_INITMEM
BD_DUMP
BD_LOAD

21 BD_DUMP

Dump the bookmark/cardmark memory to main memory.
(Not for Applications!)

COMMAND FUNCTION
INTERNAL INTERRUPT

Inputs

```
io_Command= BD_DUMP
io_Offset = ByteOffset
io_Length = NumberOfBytes
io_Data   = Buffer
```

Outputs

```
BytesMoved = io_Actual
```

Description

This command provides a means duplicating bookmark/cardmark memories. With it you can write a program that will dump the contents of your bookmark/cardmark memory to a disk file for later loading back with the BD_LOAD command. This is often helpful during application testing.

The io_Offset field can be used when the memory must be buffered in smaller chunks to conserve memory space. When using this technique, the io_Actual field can be checked to determine when the entire memory has been transferred.

Example

```
extern struct IOStdReq *IOReq;          /* IO request opened earlier */
extern long file;

#define BUF_SIZE (16 * 1024)
char Buffer[BUF_SIZE];
long offset = 0;

do
{
      IOReq->io_Command = BD_DUMP;
      IOReq->io_Offset = offset;
      IOReq->io_Length = BUF_SIZE;
      IOReq->io_Data = Buffer;
      DoIO(IOReq);
      Write(file,Buffer,IOReq->io_Actual); /* should check for file error */
      offset += BUF_SIZE;
} while (IOReq->io_Actual == BUF_SIZE);
```

Errors

Related

BD_LOAD

22 BD_LOAD

bookmark.device

COMMAND FUNCTION INTERNAL INTERRUPT

Load the bookmark/cardmark memory from main memory.
(Not for Applications!)

Inputs

```
io_Command= BD_LOAD
io_Offset = ByteOffset
io_Length = NumberOfBytes
io_Data   = Buffer
```

Outputs

```
BytesMoved = io_Actual
```

Description

This command provides a means of restoring previously dumped bookmark/cardmark memories. With it you can write a program that will load the contents of your bookmark/cardmark memory from a disk file. This is often helpful during application testing.

This command can also be used to write memory cards as part of a mass production duplication process.

The io_Offset field can be used when the memory must be buffered in smaller chunks to conserve memory space.

Example

```
extern struct IOStdReq *IOReq;             /* IO request opened earlier */
extern long file;

#define BUF_SIZE (16 * 1024)
char Buffer[BUF_SIZE];
long offset = 0;
long size;

while ((size = Read(file,Buffer,BUF_SIZE) > 0)
{
      IOReq->io_Command = BD_LOAD;
      IOReq->io_Offset = offset;
      IOReq->io_Length = size;
      IOReq->io_Data = Buffer;
      DoIO(IOReq);
      offset += BUF_SIZE;
}
```

Errors

Related

BD_DUMP

Packaging CDTV Titles

by Gail Wellington

Commodore has created a publication to establish a universal standard for the packaging and visual identity of CDTV discs. This publication (including films) is available to all signers of the CDTV license agreement. Within the publication you will find clear and concise instructions that will guide you in preparing CDTV disc packaging. Detailed diagrams and reproduceable artwork are provided for every element of your package.

Commodore's goal is to create an image that will serve as an optimum marketing environment, not only for CDTV technology, but for each and every title for years to come. We feel we have succeeded in this endeavor.

Naturally, Commodore cannot exercise any authority over independent publishers in the creation of their packaging, nor do we seek to imply any in the publication of these standards. However, all publishers of CDTV products share a responsibility to one another to participate in presenting a unified image to the public so that we may all reap the benefits of a quality image. In that spirit, we respectfully ask for your cooperation in conforming to these standards.

The following pages are taken from the introduction to the *CDTV Multimedia Disc Packaging and Graphics Standards Manual.*

Initial Steps

1. Determine the category that your CDTV title falls within and the age group of its target audience.
2. Determine the elements that will make up your packaging (see below).
3. Refer to the section of the publication that describes each element you need to create. (See the Table of Contents for help in locating each section.)
4. Follow the diagrams and templates found both in each section of the *CDTV Multimedia Disc Packaging and Graphics Standards Manual* and the envelope (labeled "SUPPLEMENT") that accompanies the manual.
5. Use the reproduction proofs provided in Section 10 of the *CDTV Multimedia Disc Packaging and Graphics Standards Manual* in creating your artwork.

What's in Your Package?

Your CDTV Disc package can consist of a combination of several different elements (i.e., long box, jewel case insert, etc.) What goes into your package will be determined by several factors, including the nature of the title, your production budget, and your aesthetic preferences. Since the packaging elements are interdependent, you should determine the make-up of your package before any production is begun.

Decisions, decisions...

Some of the decisions that you'll face in packaging a CDTV title are:

☐ Will the outer packaging be a standard long box or a window long box?

The window long box provides an economical outer package. A window is left open for the jewel case or disc caddy to show through, which then becomes your title graphics. The outer box can then be printed simply in 2-color.

The standard long box will give you more space to employ your title graphics but will most likely need to be printed in 4-color process, a more expensive method.

Commodore recommends window long box because it will help provide a unified image for CDTV titles and set them apart from conventional CD and CD+G packaging. Some markets prefer the jewel case, and it is especially appropriate for lower cost titles. On the other hand, the long box gives a greater perceived value to more costly titles and provides space for more selling copy.

☐ Will the disc itself be packaged inside a jewel case or a disc caddy?

CDTV owners will require disc caddies to play CDTV titles. However, inclusion of them with each disc will add to the cost.

Commodore recommends disc caddy. The additional value will be appreciated by customers and will translate into an increased perception of you and your products. The caddy can be packed in a sleeve so that the title is easily identified in the user's collection of titles.

☐ Will there be any additional documentation, reference cards, or other items included in the box?

If so, be sure to prepare your outer box with the proper construction to provide enough room. If the standard dimensions do not provide enough space, increasing the thickness should not affect the ability to put your package into standard store fixtures.

Materials

Commodore recommends and encourages the use of recycled paper in the making of long boxes for CDTV discs. Serious environmental concerns have caused intense criticism of long boxes in CD marketing. However, their function as both a theft deterrent and a marketing tool have kept them in use.

We feel that the only environmentally responsible solution to the problem is to use recycled paper in our long boxes.

We further encourage that, when it applies, you include an appropriate statement to that effect on the back of your box. Example:

Made from recycled paper.

or, if applicable,

Made from 100% recycled paper.

Categories and Age Groups

CDTV applications will span a broad spectrum of interests and audiences. Because of this diversity, and to help introduce consumers to the world of CDTV, Commodore has devised a system of seven categories to help explain each title and identify its likely audience.

Choosing a specific category for your title may seem difficult and limiting at times. By nature, many CDTV applications cross common borders of specialization. However, we strongly urge that you choose one and only one category and one age group that best applies to your title. Remember, this is only a supplemental aid for customers. Your title graphics and description will tell the real story, so we suggest that you exercise that opportunity to present the features and benefits of your application.

The categories are as follows:

Category	Description
Arts & Leisure	Non-game activities for spare time including hobbies, how-to, etc. Examples: gardening, home repair, video titling, performing and visual arts references.
Education	Learn-to or practice basic education skills. Also academic tools. Examples: learn-to-read, simulations of physics experiments, foreign language tutors.
Entertainment	Games of all types (action, thinking, board game simulations, etc.) Examples: Flight simulators, arcade and adventure games, chess.
Music	All types of music-oriented titles, including participatory and reference Examples: Learn to play recorder, classical music reference works, play along, MIDI,CDTV-specific CD+G.
Periodicals	Reference or other works released on a regular schedule. Examples: Catalogs, magazines, directories.
Productivity	Traditional computer applications. Examples: Word processors, spreadsheets, databases, home accounting.
Reference	Sources of information intended to be looked up or explored. Examples: dictionaries, encyclopedias, atlases, "coffee table books".

Age Identification

The age of your target audience should be described in one of the following manners. Again, please choose only one description.

For children ages _____ to _____
For teens and adults
For all ages
For adults

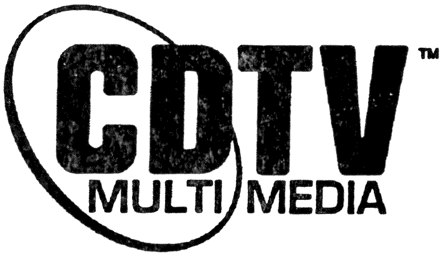
CDTV MULTIMEDIA Logo

The CDTV MULTIMEDIA logo, based on the original CDTV logo, is the foundation for this entire identity program. Like these guidelines, it is the result of a long, intense, creative process to develop a trademark that will project a unique, positive identity for these products.

In the use of the CDTV MULTIMEDIA logo, we ask that you maintain strict adherence to the instructions in the *CDTV Multimedia Disc Packaging and Graphics Standards Manual.*

The CDTV logo consists of the CDTV logotype, the disc graphic, the type "MULTIMEDIA" and the "TM" symbol. These elements are NEVER to be separated, used independently of each other, or in any way altered from the form shown below except as specified in the *CDTV Multimedia Disc Packaging and Graphics Standards Manual.*

THE LOGO IS NEVER TO BE REDRAWN OR RE-CREATED. IT MUST ALWAYS BE REPRODUCED PHOTOMECHANICALLY FROM THE REPRODUCTION PROOFS IN THE MANUAL.



DO NOT REPRODUCE FROM THIS PAGE

Design Elements

Your design should consist of the following elements:

A. TOP BARS: A band of 100% black .5" tall along the top edge of the front and back covers. The top panel should also print 100% black.

B. CDTV MULTIMEDIA LOGO: Centered within both top bars in REVERSE form. Follow examples for size and position. (See manual for reproduction proofs.)

C. TYPOGRAPHY:

CATEGORY: Your title category set in 12 pt. Helvetica Regular, upper and lower case. Reverse white[1] from color bar. Place as shown. (See page 3 to find which category applies to your title.)

AGE GROUP: Set in 10 pt. Helvetica Regular, upper and lower case. Reverse white[1] from color bar. Place as shown. (See page 3 for correct age group phrasing.)

CDTV: The letters CDTV set in 10 pt. Helvetica Regular, all caps. Reverse white[1] from top panels and color bars; surprint 100% black on bottom panel.

[1] Except for Education category which surprints 100% black.

D. FOR PUBLISHER USE: The areas between the top bar and color bar are free for you to use for title graphics and publisher information.

E. COLOR BARS: .5" bands of color at bottom edge of front and back panels identifying the title category.

Category	Pantone Ink	Process Equivalent
Arts & Leisure	PMS 299	20%M, 100%C
Education	PMS 108	100%Y
Entertainment	PMS 032	100%Y, 100%M
Music	PMS 265	70%M, 60%C
Periodicals	PMS 354	100%Y, 100%C
Productivity	PMS 021	90%Y, 60%M
Reference	PMS 160	100%Y, 60%M,, 40%K

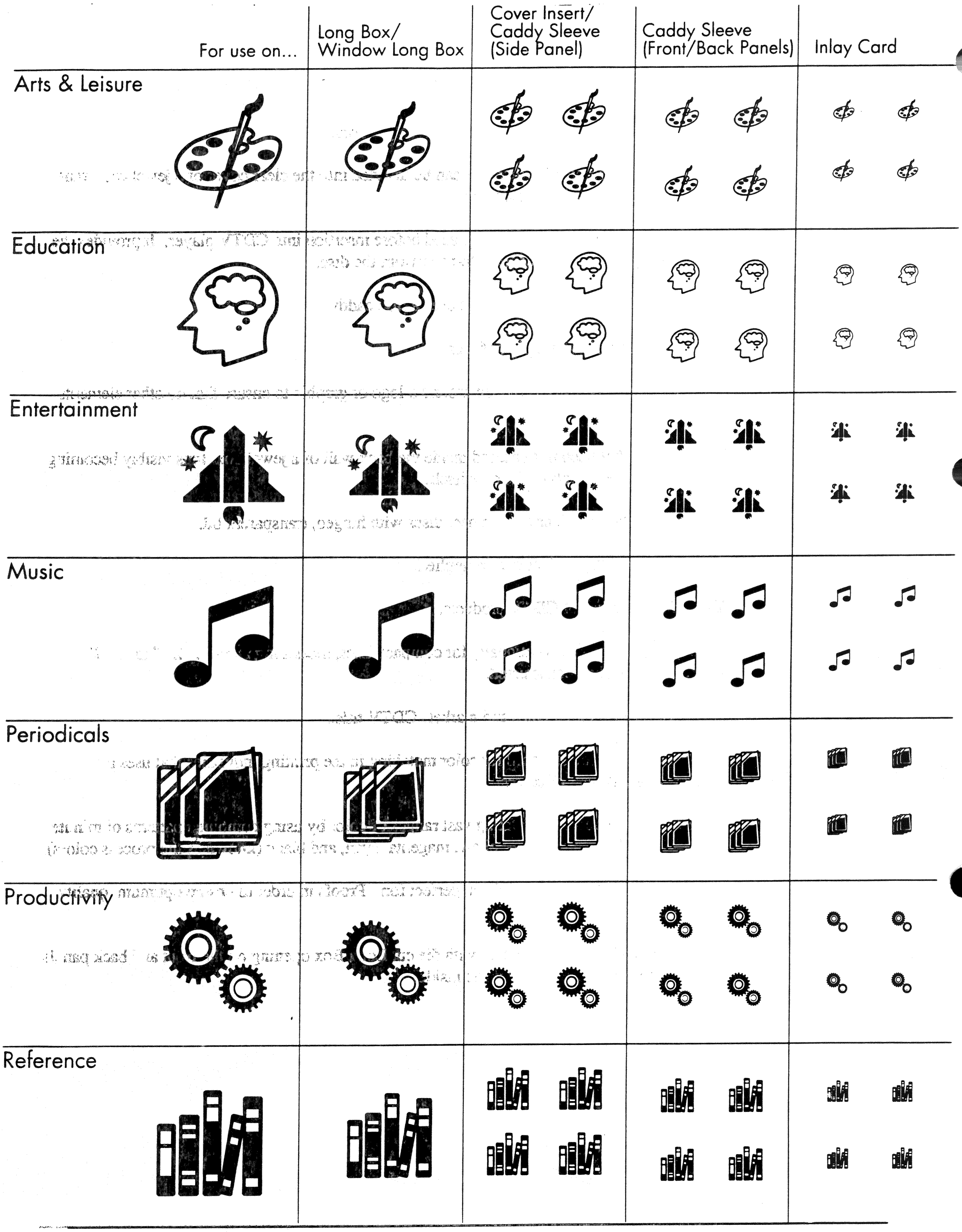
F. CATEGORY SYMBOL: See next page for examples and manual for reproduction proofs. Symbol reverses white from color bars[1].

---

[1] Except for Education category which surprints 100% black.

CATEGORY SYMBOLS

DO NOT REPRODUCE FROM THIS PAGE


For use on...
Long Box/ Window Long Box
Cover Insert/ Caddy Sleeve (Side Panel)
Caddy Sleeve (Front/Back Panels)
Inlay Card
Arts & Leisure
Education
Entertainment
Music
Periodicals
Productivity
Reference

Glossary

Bleed	Extending graphics beyond the edge as a safety margin.
Booklet	Pamphlet that accompanies discs. It can be inserted into the clear cover of a jewel case, thus becoming the cover of the case itself.
Disc Caddy	Cartridge into which discs must be placed before insertion into CDTV player. It provides the stability necessary for correct data transfer from the disc.
Caddy Sleeve	Paperboard box or wrapping that encloses a disc caddy.
Category	CDTV titles of a similar topic or focus.
Control Space	A margin of empty space provided around a logo or graphic to ensure that no other elements "crowd" it.
Inlay Card	Printed sheet that becomes encased inside the back wall of a jewel box, thus visibly becoming the back and side panels of the case itself.
Jewel Case	Standard plastic container for compact discs with hinged, transparent lid.
Live Area	Safe area in which graphics can be applied.
Logo	The official symbol of CDTV products.
Long Box	Outer package used conventionally for compact discs, measuring approx. 12.3" tall, 5.7" wide, .5" deep when constructed.
Publisher	Any organization that creates and markets CDTV titles.
Pantone/PMS	Universally accepted standard for color matching in the printing industries that uses a numbered assortment of colors.
Process	The printing method of simulating vast ranges of color by using combined patterns of minute dots of 4 standard color inks--yellow, magenta, cyan, and black (known as the process colors).
Reproduction	Artwork that is provided in a clean, perfect form Proofs in order to ensure optimum quality in reproduction.
Window	Modification of standard long box with die cut Long Box opening on the front and back panels allowing a view of the disc package inside.

Distributing CDTV Titles Worldwide

DIFFERENT MECHANISMS FOR DIFFERENT MARKETS

International Vertical Markets:

- ☐ Library markets
- ☐ Corporate markets

Options for vertical markets:

1. Direct sales force
2. Specialist distributors, i.e. EBSCO (USA), Optech (UK)
3. Bundling
4. Better margins for publishers as there is one less step in the distribution chain (say, discount at 30% to 40%).

International Mass Markets:

- ☐ The High Street, the shopping mall, the corner shop

The mechanism of publishing software for mass markets:

1. Publisher. Must do advertising and public relations to create consumer demand. Sells to:
2. Distributor. Must promote to retail sector. Sells to:
3. Individual retail outlets. Who sells to consumer.

Note 1: Mass merchandisers (chains of retail outlets) sometimes buy directly from publishers, sometimes through distributors

Note 2: There are significant national differences in this pattern.

THE TRADITIONAL STEPS TO ENTER A NEW TERRITORY

1. Licensing to national publishers

2. Affiliate label arrangements

3. National publishing

New opportunities:

New outlets (non-traditional computer outlets)
CD (music) outlets
Book stores

RETAIL PRICING PROBLEM OF INTERNATIONAL PUBLISHING

1. Each national market has its own "felt fair" retail price for a particular type of product.

2. Exchange rates can fluctuate by 50% or more during the life of a product.

3. Allowance needs to be made for shipping costs, duty, VAT.

4. Discounting higher than vertical markets (50% to 60%).

Resulting problem:

If you sell at the same absolute price to distributors worldwide, margin structure is destroyed. For example,

Publisher sells at $20	$1.8:Sterling 1	$1.3:Sterling 1
Consumer buys	22.22 Sterling ($40)	22.22 Sterling ($29)

Solution:

☐ Sell into each territory at the standard discount on the local recommended retail price.

☐ Discourage "grey marketing" by relevant means.